

FESA - FACULDADE ENGENHEIRO SALVADOR ARENA

ENGENHARIA DE COMPUTAÇÃO

ENZO BRITO ALVES DE OLIVEIRA – RA: 082220040

ERIKSON VIEIRA QUEIROZ – RA: 082220021

GUILHERME ALVES BARBOSA – RA: 082220014

HEITOR SANTOS FERREIRA – RA: 081230042

TAINARA DO NASCIMENTO CASIMIRO – RA: 082220011

WILLIAM SANTIM – RA:082220033

**CÁLCULO E SIMULAÇÃO DE UM LANÇAMENTO BALÍSTICO COM ALVO
MÓVEL**

SÃO BERNARDO DO CAMPO

2023

- INTRODUÇÃO

O Projeto consiste em um trabalho multidisciplinar, envolvendo quatro disciplinas cursadas no terceiro semestre do curso de Engenharia de Computação na Faculdade Engenheiro Salvador Arena. As disciplinas envolvidas são: Banco de Dados II, Cálculo Avançado, Física Geral e Experimental II e Programação Orientada a Objetos.

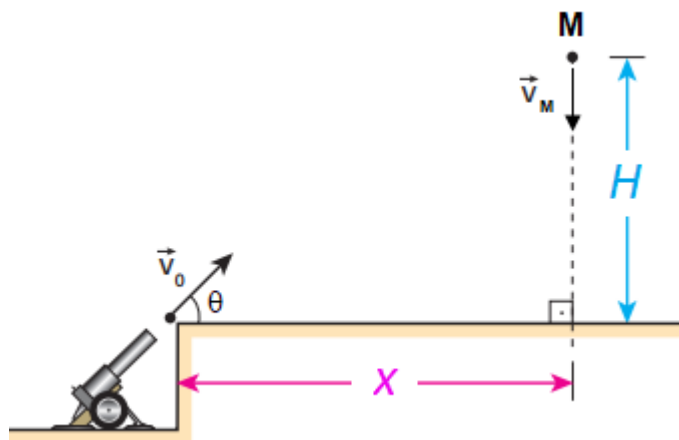
Sendo assim, cada disciplina propôs uma necessidade na situação problema, que visa aplicar na prática as teorias ensinadas em sala de aula. A seguir, segue um breve detalhamento sobre o que foi pedido:

- Banco de Dados II: disponibilização de todo o banco de dados da aplicação através de Stored Procedure. Todo o CRUD deverá ser feito através de chamadas a SP no SQL-Server, assim como eventuais consultas de dados no sistema. É permitido o uso de tabelas temporárias e deverá existir ao menos 3 triggers dentro do sistema.
- Cálculo Avançado: Como a utilização de bibliotecas matemáticas não é permitida, faz-se necessária a utilização da teoria vista na matéria (séries de Taylor e MacLaurin) para cálculo dos valores de senos e cossenos dos ângulos de interesse da situação problema;
- Física Geral e Experimental II: Lançamento de projétil. Na horizontal (eixo x), o projétil descreve um movimento retilíneo uniforme (MRU), cuja equação de posição x em função do tempo t é: $x(t) = x_0 + v \cdot t$ e na vertical (eixo y) o projeto descreve um movimento retilíneo uniformemente acelerado (MRUA), cuja equação da posição y em função do tempo t é $y(t) = y_0 + v_{y0} t + \frac{1}{2} a_y t^2$.
Com relação a altura máxima, ocorre conforme a equação de Torricelli.
- Programação Orientada a Objetos: Programação em C#, conceitos de orientação a objetos (classes, atributos, objetos, abstração, encapsulamento, herança, polimorfismo), utilização de UML para representação e relacionamento entre classes por meio de elaboração de diagrama de classes.

- FORMULAÇÃO DO PROBLEMA E OBTENÇÃO DE EQUAÇÕES

O objetivo principal desse projeto é resolver um problema físico. Um meteoro adentra a atmosfera e ao aproximar-se do solo, deverá ser abatido. A suposição é de que sua trajetória é vertical e no instante $t_0 = 0$ segundos, o meteoro está com velocidade constante V_M e uma altura H em relação ao solo.

Para tentar abater o alvo, um projétil é lançado obliquamente em relação ao solo (em um terreno plano e horizontal). A resistência do ar é desprezível e a aceleração gravitacional local tem intensidade $g = 9,8 \text{ m/s}^2$.



Ao lançar um projétil, observa-se que a sua trajetória é uma curva. Essa curva pode ser descrita através da composição de 2 movimentos: Movimento Retilíneo Uniforme (MRU) na horizontal e Movimento Retilíneo Uniformemente Acelerado (MRUA) na vertical, sujeito a aceleração da gravidade.

Considera-se o lançamento de um projétil por um canhão inclinado de um ângulo θ , lançado do repouso, sobre a superfície da Terra, a partir da origem de um sistema de coordenadas xy . O projétil abandona o canhão com uma velocidade inicial v_0 e descreve uma trajetória curvilínea até atingir o solo a uma distância igual a x .

Componentes de v_0 : $v_{0x} = v_0 \cdot \cos\theta$ e $v_{0y} = v_0 \cdot \sin\theta$

Na horizontal, deve-se ter: $v_{0x} = \frac{x}{t} \rightarrow t = \frac{x}{v_0 \cdot \cos\theta}$ (I)

O intervalo de tempo expresso pela relação (I) é o necessário para abater o meteoro/alvo.

Para que o encontro entre projétil e alvo se efetive, é necessário que ambos passem pela mesma coordenada vertical y no instante t, dado por (I). Portanto:

$$\text{Meteoro: } y_M = H - v_M \cdot t \quad (\text{II})$$

$$\text{Projétil: } y_P = v_{0y} \cdot t - \frac{g}{2} t^2 \quad \rightarrow \quad y_P = (v_0 \cdot \text{sen}\theta)t - 4,9t^2 \quad (\text{III})$$

$$\text{Igualando-se (II) e (III), vem: } y_M = H - v_M \cdot t = (v_0 \cdot \text{sen}\theta)t - 4,9t^2 \quad (\text{IV})$$

$$\text{Substituindo-se (I) em (IV), vem: } H - \frac{x \cdot v_M}{v_0 \cdot \cos\theta} = x \cdot \text{tg}\theta - \frac{4,9 x^2}{v_0^2 \cdot \cos^2\theta}$$

Ao multiplicar esta última expressão por v_0^2 , obtém-se:

$$Hv_0^2 - \frac{x \cdot v_M \cdot v_0}{\cos\theta} = v_0^2(x \cdot \text{tg}\theta) - \frac{4,9 x^2}{\cos^2\theta}$$

Reagrupando os termos, obtemos a seguinte equação de 2º grau:

$$(x \cdot \text{tg}\theta - H)v_0^2 + \left(\frac{x \cdot v_M}{\cos\theta}\right)v_0 - \left(\frac{4,9 x^2}{\cos^2\theta}\right) = 0$$

Uma equação de 2º grau tem como formulação básica $Ax^2 + Bx + C$, da equação obtida temos:

$$A = x \cdot \text{tg}\theta - H$$

$$B = \frac{x \cdot v_M}{\cos\theta}$$

$$C = \frac{4,9 x^2}{\cos^2\theta}$$

Ao calcular o delta podemos achar os valores das raízes da equação. O valor positivo é a velocidade de v_0 .

A partir daí faz-se necessário calcular a coordenada Y_M do meteoro (conforme equação II), coordenada Y_P do projétil (conforme equação III) e coordenada x (conforme equação I). A comparação dos valores das coordenadas no eixo y do projétil e do meteoro deve ser feita, quando elas forem iguais significa que o projétil atingiu o meteoro, ou seja o objetivo foi atingido e deve-se notar em qual instante de tempo t e em qual coordenada de posicionamento no eixo x esse encontro ocorreu.

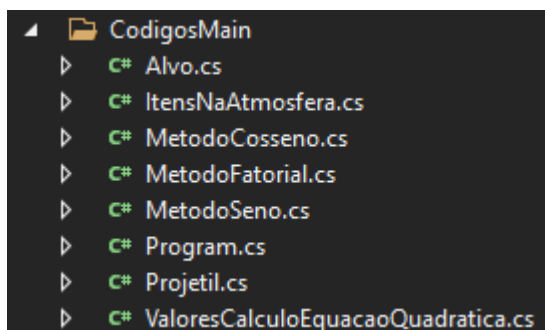
- DESENVOLVIMENTO DO SOFTWARE

Com base na formulação do problema e obtenção de equações, temos as condições de contorno e etapas que o software deverá atender (solicitações do cliente), para que o desenvolvimento seja feito baseado nas necessidades listadas abaixo:

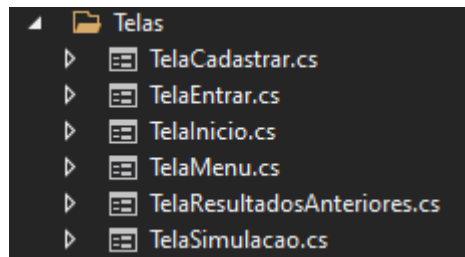
- 1) Perguntar ao usuário do programa se ele gostaria de recuperar os resultados obtidos anteriormente para um dado valor de θ , (já existente no banco de dados) ou incluir um novo valor. Neste caso, o software deverá solicitar que o usuário entre com um valor de θ ;
- 2) Fornecer o(s) valor(es) da velocidade inicial do projétil, para que o objetivo seja cumprido;
- 3) Encontrar o intervalo de tempo gasto desde o lançamento do projétil até atingir o alvo;
- 4) Informar se o alvo é atingido quando o projétil está em movimento ascendente ou descendente
- 5) Ilustrar a evolução das coordenadas horizontal (x) e vertical (y) do projétil e do meteoro ao longo do tempo (t).

Sabendo as necessidades do cliente, podemos listar algumas etapas chave no desenvolvimento do programa a seguir:

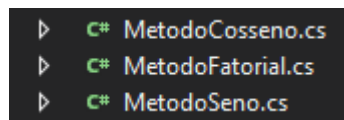
- Modelamento dos objetos (alvo e projétil), variáveis e atributos que serão utilizados nos cálculos da resolução da situação problema, criação de construtores, métodos e a interação deles com as telas do Forms, assim como a inserção dos dados no banco de dados:



- Criação das telas do Forms, visando uma melhor interface visual entre o usuário e o programa:

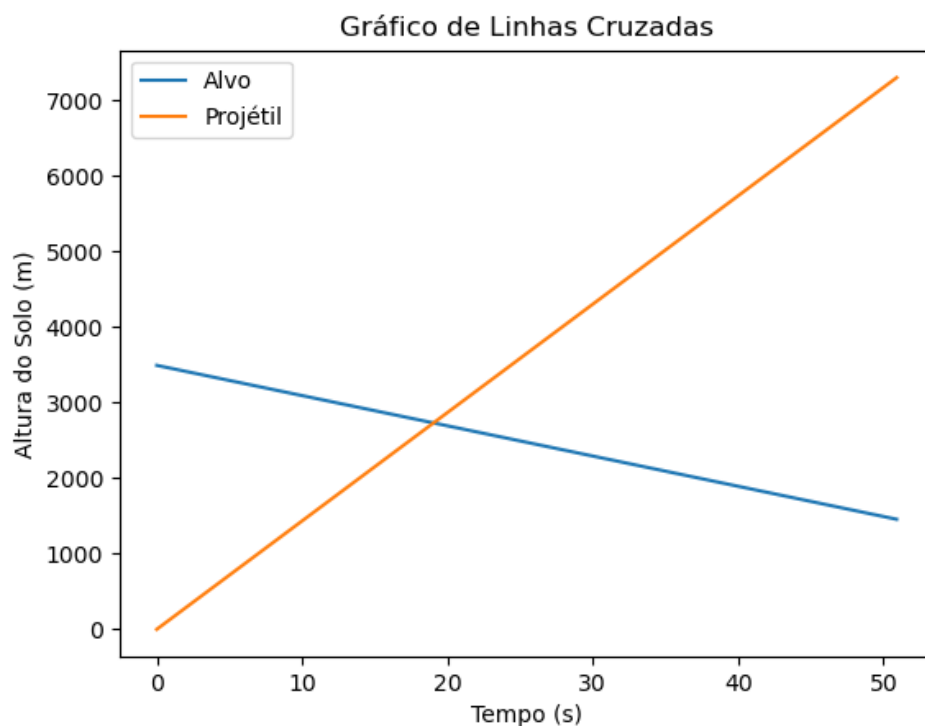


- Criação das funções para os cálculos de seno e cosseno, utilizando como base a teoria aprendida na disciplina de cálculo avançado (não é permitido a utilização de bibliotecas matemáticas), assim como a sua aplicação nos cálculos durante a execução do programa;



- Formulação dos cálculos conforme teoria física da situação problema proposta

- Formulação da parte de geração do gráfico, ilustrando a evolução nas coordenadas x e y do alvo e do projétil (ao longo do tempo), inclusive o momento de encontro



- Criação do banco de dados juntamente com suas tabelas, arquitetura, modelos de inserção, validações por SP, triggers para criação de interações com o usuário e modelo de relacionamento entre as tabelas;

▲	📁	Tabelas
▲	📄	alvo
	🔑	id_alvo
	📄	altura_inicial_alvo
	📄	distancia_canhao_alvo
	📄	velocidade_inicial_alvo
	📄	angulo_maximo_acerto_alvo
	📄	tempo_ate_attingir_solo_alvo
▲	📄	projatil
	🔑	id_projatil
	📄	angulo_trajetoria_projatil
	📄	velocidade_inicial1_projatil
	📄	velocidade_inicial2_projatil
	📄	velocidade_usada_projatil
	📄	tempo_de_voo_projatil
	📄	tipo_de_movimento_projatil
	📄	existencia_resolucao_real_angulo_fornecido
▲	📄	simulacoes
	🔑	id_simulacao
	📄	id_projatil
	📄	id_alvo
	📄	angulo_trajetoria_projatil
	📄	velocidade_inicial1_projatil
	📄	velocidade_inicial2_projatil
	📄	velocidade_usada_projatil
	📄	tempo_de_voo_projatil
	📄	tipo_movimento_projatil
	📄	existencia_resolucao_real_angulo_fornecido
	📄	altura_inicial_alvo
	📄	distancia_canhao_alvo
	📄	velocidade_inicial_alvo
	📄	angulo_maximo_acerto_alvo
	📄	tempo_ate_attingir_solo_alvo

▲	📄	usuarios
	🔑	id_usuario
	📄	nome_usuario
	📄	senha_usuario
	⚡	VerificaExistenciaUnicaUsuario
	⚡	VerificaNomeUsuario
	⚡	VerificaSenhaUsuario

- DESCRIÇÃO DETALHADA DO SOFTWARE

A seguir temos a descrição detalhada do software, mostrando através de prints das telas o desenvolvimento de software.

- ALVO:

```
//Define o namespace em que o código está inserido
namespace PPL_Main.CodigosMain
{
    //Declarando a classe/objeto alvo como publico
    public class Alvo
    {
        //Declarando atributos
        private double alturaInicialAlvo;
        private double distanciaDoCanhaoAlvo;
        private double tempoAteAtingirSoloAlvo;
        private double velocidadeAlvo;
        private double anguloMaximo;
        private double posicaoEmYAlvo;

        //Gerando encapsulamento para os atributos necessários
        //Gerando encapsulamento para o atributo "alturaInicialAlvo"
        public double AlturaInicialAlvo
        {
            get { return alturaInicialAlvo; }
            set { alturaInicialAlvo = value; }
        }

        //Gerando encapsulamento para o atributo "distanciaDoCanhaoAlvo"
        public double DistanciaDoCanhaoAlvo
        {
            get { return distanciaDoCanhaoAlvo; }
            set { distanciaDoCanhaoAlvo = value; }
        }

        //Gerando encapsulamento para o atributo "velocidadeAlvo"
        public double VelocidadeAlvo
        {
            get { return velocidadeAlvo; }
            set { velocidadeAlvo = value; }
        }

        //Gerando encapsulamento para o atributo "tempoAteAtingirSoloAlvo"
        public double TempoAteAtingirSoloAlvo
        {
            get { return tempoAteAtingirSoloAlvo; }
            set { tempoAteAtingirSoloAlvo = value; }
        }
    }
}
```

```
//Gerando encapsulamento para o atributo "anguloMaximo"
public double AnguloMaximo
{
    get { return anguloMaximo; }
    set { anguloMaximo = value; }
}

//Gerando encapsulamento para o atributo "posicaoEmYAlvo"
public double PosicaoEmYAlvo
{
    get { return posicaoEmYAlvo; }
    set { posicaoEmYAlvo = value; }
}

//Criando métodos da classe para melhorar processamento futuros
//Criando método que realiza calculo do tempo que decorre até que o alvo atinja o solo
public void CalculaTempoAteAtingirSoloAlvo()
{
    //Atribui o atributo antes declarado "TempoAteAtingirSoloAlvo" com o valor do resultado da divisão entre a altura inicial do alvo
    //("AlturaInicialAlvo") e a velocidade inicial do alvo ("VelocidadeAlvo")
    TempoAteAtingirSoloAlvo = (AlturaInicialAlvo / VelocidadeAlvo);
    //Realiza o arredondamento do valor no caso de valores do resultado gerarem mais de duas vírgulas após a casa decimal comum
    TempoAteAtingirSoloAlvo = Math.Round(TempoAteAtingirSoloAlvo, 2);
}

//Criando método que realiza o calculo do angulo máximo de lançamento que o projétil precisa ter para atingir o alvo com sucesso
public void CalculaAnguloMaximo()
{
    //Atribui o atributo antes declarado "AnguloMaximo" com o valor do resultado da tangente do valor entre a altura inicial do alvo
    //("AlturaInicialAlvo") e a distância do alvo até o canhão ("DistanciaDoCanhaoAlvo")
    AnguloMaximo = (Math.Atan(AlturaInicialAlvo / DistanciaDoCanhaoAlvo)) * 180 / Math.PI;
    //Realiza o arredondamento do valor obtido no caso de valores do resultado gerarem mais de duas vírgulas após a casa decimal comum
    AnguloMaximo = Math.Round(AnguloMaximo, 2);
}
```

- ITENS NA ATMOSFERA

```
namespace PPL_Main.CodigosMain
{
    0 referências
    public class ItensNaAtmosfera
    {
        1 referência
        public double valorAltura { get; protected set; }
        protected double valorGraviade = 10;
        protected double velocidadeInicialEmVitensNaAtmosfera;

        0 referências
        public virtual void DeterminaAltura(double tempoDecorrido)
        {
            valorAltura = (velocidadeInicialEmVitensNaAtmosfera * tempoDecorrido) - (valorGraviade / 2) * Math.Pow(tempoDecorrido, 2);
        }
    }
}
```

- MÉTODO COSSENO

```
//Evoca a biblioteca "System", referenciando a suas funções genéricas
using System;

//Define o namespace em que o código está inserido
namespace PPL_Main.CodigosMain
{
    //Referenciando a classe de "MetodoCosseno" como pública
    3 referências
    public class MetodoCosseno
    {
        //criando função principal da classe que realiza o cálculo do Cosseno

        //A função é declarada com um meio de público e estático, recebendo um valor em meio 'double' que é chamado de 'x'.
        //Durante o uso dessa função ao decorrer do calculo, o valor de 'x' é geralmente recebido como o valor do ângulo em radianos a ser calculado
        1 referência
        public static double CalculaCosseno(double x)
        {
            //Sugere uma precisão desejada, elevando o número 10 a uma visão de -9 vezes.
            double precisaoDesejada = Math.Pow(10, -9);

            //Definie uma variavel como 'bool' para verificar o atingimento da precisão desejada,
            //essa variavel é chamada ao decorrer do código quando a precisão alcançada é encontrada

            bool precisaoDesejadaAlcancada = false;
            //Define a variavel 'k' como tipo inteiro, a variavel 'k' é o número de iterações a ser realizada, inicialmente recebe o valor de '0'
            int k;
            k = 0;

            //Declara o valor da função que realiza a 'descoberta' do valor calculado como zero em uma primeira instancia
            double valorFuncaoF = 0;

            //Declara uma variavel que irá receber o valor da raiz da função
            double raizFuncaoF;

            //Cria um laço de repetição 'while' que tem sua continuação mantida, determinada pelo atingimento da precisão desejada
            while (!precisaoDesejadaAlcancada)
            {
                //Realiza a criação de diversas determinações de calculo
                //O primeiro 'if' realiza a verificação do valor de 'k', onde, caso o mesmo esteja zerado, recebe a interação de um valor '1' (um).
                if (k == 0)
                {
                    valorFuncaoF += 1; //Adiciona o valor de '1' (um) a váriavle que recebe o valor da função
                }
                //Essa condição julga o valor de k entre divisões de sobra e recebimento de determinantes diferentes de zero
                else if (k >= 1 && k % 2 == 0 && k % 4 != 0)
                {
                    //O valor da função recebe então a interação de um valor determinado pela conta matemática aplicada, que nesse caso é: o valor de -1,
                    //vezes a elevada do valor de 'x' pelo número de iterações correspondente no momento dividido pela fatorial do valor das iterações correspondentes.
                    //O valor da fatorial é calculado evocando a classe "MetodoFatorial" e a função "CalculaFatorial"
                    valorFuncaoF += (-1) * Math.Pow(x, k) / MetodoFatorial.CalculaFatorial(k);
                }
            }
        }
    }
}
```

```

//Essa condição julga o valor de k entre as sobras dos valores obtidos pelo mesmo após cada iteração
else if (k >= 1 && k % 4 == 0)
{
    //O valor da função recebe então a iteração de um valor determinado pela conta matemática aplicada,
    //que nesse caso é: a elevada do valor de 'x' pelo número de iterações correspondente no momento dividido pela fatorial do valor das iterações correspondentes.
    //O valor da fatorial é calculado evocando a classe "MetodoFatorial" e a função "CalculaFatorial"
}
valorFuncaoF += (Math.Pow(x, k)) / MetodoFatorial.CalculaFatorial(k);

//Determina um valor para a variável "raizFuncaoF" declarada anteriormente

//O valor da variável é determinado pela elevação do valor em modulo de 'x'
//pelo número de iterações mais um, dividido pela fatorial do valor da iteração mais um.
//O valor da fatorial é calculado evocando a classe "MetodoFatorial" e a função "CalculaFatorial"
raizFuncaoF = (Math.Pow(Math.Abs(x), (k + 1)) / MetodoFatorial.CalculaFatorial(k + 1));

//Realiza a verificação perante o julgamento de que caso o valor da raiz da função calculada seja menor ou igual o valor da precisão desejada,
//a variável de nível de julgamento 'verdadeiro' ou 'falso' ('bool') criada anteriormente, é definida como verdadeira
if (raizFuncaoF <= precisaoDesejada)
{
    precisaoDesejadaAlcancada = true; //Define a variável do tipo 'bool' como true, nesse momento, o loop while é interrompido
}
//Caso a verificação realizada no 'if' falhe, é adiciona mais uma iteração a variável 'k'
else
{
    k += 1;
}

//Declara o retorno do valor da função em F como valor da função
return valorFuncaoF;
}
}

```

- MÉTODO SENO:

```

using System;

//Define o namespace em que o código está inserido
namespace PPL_Main.CodigosMain
{
    //Declarando o nome da classe como MetodoSeno e declarando ela como pública
    2 referências
    public class MetodoSeno
    {
        //Criando método principal da classe, chamando de "CalcularSeno" e declarando o mesmo como público
        2 referências
        public static double CalcularSeno(double x) //Declara do valor da função como 'double', sendo assim, a mesma retornará um valor nesse tipo de dado.
        {
            //Define um valor de tipo 'double' como evocador da função, chamando o mesmo de 'x'.
            //Ao decorrer do código e do uso da função em outros trechos do software,
            //o valor de 'x' recebe o valor de ângulos em radianos e desenvolve o valor do mesmo.

            double precisaoDesejada = Math.Pow(10, -9); //Sugere uma precisão desejada, elevando o número 10 a uma visão de -9 vezes.

            bool precisaoDesejadaAlcancada = false; //Define uma variável como 'bool' para verificar o atingimento da precisão desejada,
            //essa variável é chamada ao decorrer do código quando a precisão alcançada é encontrada

            //Define a variável 'k' como tipo inteiro, a variável 'k' é o número de iterações a ser realizada, inicialmente recebe o valor de '0'
            int k;
            k = 0;

            //Declara o valor da função que realiza a 'descoberta' do valor calculado como zero em uma primeira instancia
            double valorFuncaoF = 0;

            //Declara uma variável que irá receber o valor da raiz da função
            double raizFuncaoF;

            // Define uma variável que irá receber o valor de iterações, definindo ela como tipo inteiro e recebendo um valor inicial de -1
            int iteracao = -1;

            //Cria um laço de repetição while que tem como condição de sua continuidade de execução o alcance da precisão desejada
            while (!precisaoDesejadaAlcancada)
            {
                //Cria a primeira verificação no iterador 'k', onde caso o mesmo seja igual a 1 (um), é adicionado uma interação do valor declarado em 'x' para o valor da função
                if (k == 1)
                {
                    valorFuncaoF += x;
                }
            }
        }
    }
}

```

```

//Realiza uma alternativa a verificação inicial, onde caso o valor do iterador 'k' tenha um valor de resto de divisão por dois diferente de zero,
//é adicionada um valor na função de calculo.
else if (k % 2 != 0)
{
    //Itera um valor no valor da função onde a variável recebe o valor da iteração atual multiplicado pelo valor do ângulo elevado pelo número da iteração,
    //dividido pela fatorial do número da iteração, o valor da fatorial é calculado evocando a classe "MetodoFatorial" e a função "CalculaFatorial"
    valorFuncaoF += iteracao * Math.Pow(x, k) / MetodoFatorial.CalculaFatorial(k);
    iteracao *= -1; //Eleva em modelo de iteração por menos um
}

//Define o valor da raiz da função F como 1 (um) dividido pela fatorial (O valor da fatorial é calculado evocando a classe "MetodoFatorial"
//e a função "CalculaFatorial") do número da iteração mais um,
//multiplicando esse resultado pelo valor de elevação do módulo do ângulo em radianos fornecido pelo número de iteração mais 1 (um)
raizFuncaoF = (1 / MetodoFatorial.CalculaFatorial(k + 1)) * Math.Pow(Math.Abs(x), k + 1);

//Realiza a construção de um segundo modelon de verificação.
//Esse segundo julga o atendimento da condição em que a raiz da função calculada seja menor ou igual ao valor da precisão desejada
if (raizFuncaoF <= precisaoDesejada)
{
    precisaoDesejadaAlcancada = true; //caso a verificação seja verdadeira, realiza a alteração da variável que julga a continuidade do laço 'while' para 'true',
    //interrompendo o funcionamento do mesmo.
}

//caso a verificação não seja atendida, realiza a iteração do valor de 1 (um) do valor do iterador
else
{
    k += 1;
}
}

//Retorna o valor da função calculada
return valorFuncaoF;
}
}

```

- MÉTODO FATORIAL

```

//Evoca a biblioteca "System", referenciando a suas funções gerênicas
using System;

//Define o namespace em que o código está inserido
namespace PPL_Main.CodigosMain
{
    //Define a classe "MetodoFatorial" como publica
    5 referências
    public class MetodoFatorial
    {
        //Declarando o método de calcular a Fatorial
        5 referências
        public static double CalculaFatorial(int n) //Nesse meio, o método é declarado como público e seu tipo de retorno é definido como 'double',
        //recebendo um valor do tipo inteiro chamado de 'n' que durante o decorrer do código e a
        //aplicação desse método ao decorrer da construção do software em geral,
        //será responsável por determinar o número de valores a ser incluído na fatorial
        {
            //Determina uma variável de nome 'valorFatorial' como inteiro e atribui um valor inicial de 1 (um) para ela
            int valorFatorial = 1;

            //Verifica o valor do número de fatoriais a ser realizado. Caso o valor seja maior que 0 (zero)
            if (n > 0)
            {
                //Cria um laço de repetição/verificação, onde o número determinado dentro da função recebe o valor de 'n',
                //julgando que ele seja maior que zero e regresse um em sua sequencia natural
                for (int i = n; i > 0; i--)
                {
                    //Adiciona uma multiplicação dinamica e iterativa a variável de 'valorFatorial'
                    valorFatorial *= i;
                }
            }

            //Retorna o valor atribuído a variável 'valorFatorial' em sua condição final
            return valorFatorial;
        }
    }
}

```

- PROGRAM

```
//Declarando as bibliotecas que serão utilizadas
using System;
using System.Data.SqlClient;
using System.Windows.Forms;

//Criando a parte/namespace principal do programa
namespace PPL_Main
{
    //Criando a parte da classe executora do programa principal
    10 referencias
    public class Program
    {
        //Cria uma string principal que recebe o grau de confienciabilidade como público, e o valor do endereçamento do banco de dados.
        //ESSE VALOR DEVE SER ALTERADO CONFORME A MÁQUINA NO QUAL O SOFTWARE ESTÁ SENDO EXECUTADO
        public string enderecamentoBancoDeDados = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachobFilename=D:\\Meitor\\FESA\\PBL\\PPL_Main\\PPL_Main\\PBLDB.mdf;Integrated Security=True";

        /// <summary>
        /// Ponto de entrada principal para o aplicativo.
        /// </summary>
        [STAThread]

        //Realiza a criação da execução principal do software
        0 referencias
        static void Main()
        {
            Application.EnableVisualStyles(); //Habilita os visuais do WindowsForms para serem inicializados corretamente

            //Desabilita a compatibilidade variável específica de renderização de visuais do WindowsForms.
            //Isso é desabilitado por padrão para que não ocorra problema de compatibilidade específica causada pela execução de um mesmo software em diferentes computadores
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Telas.TelaBemvindo()); //Executa a aplicação, abrindo por principio a Tela com o nome de 'TelaBemvindo'
        }
    }
}
```

- PROJÉTIL

```
2 referências
public class Projétil
{
    public double gravidade = 9.81;
    private double velocidadeInicial1Projétil;
    private double velocidadeInicial2Projétil;
    private double velocidadeUsada;
    private double anguloLancamentoProjétil;
    private double tempoVoo;
    private string tipoMovimento;
    private string resolucaoRealParaOAngulo;
    private double velocidadeInicialEmXProjétil;
    private double velocidadeInicialEmYProjétil;
    private double posicaoEmYProjétil;

    6 referências
    public double VelocidadeInicial1Projétil
    {
        get { return velocidadeInicial1Projétil; }
        set { velocidadeInicial1Projétil = value; }
    }

    5 referências
    public double VelocidadeInicial2Projétil
    {
        get { return velocidadeInicial2Projétil; }
        set { velocidadeInicial2Projétil = value; }
    }

    7 referências
    public double VelocidadeUsada
    {
        get { return velocidadeUsada; }
        set { velocidadeUsada = value; }
    }

    13 referências
    public double AnguloLancamentoProjétil
    {
        get { return anguloLancamentoProjétil; }
        set { anguloLancamentoProjétil = value; }
    }

    5 referências
    public string TipoMovimento
    {
        get { return tipoMovimento; }
        set { tipoMovimento = value; }
    }
}
```

```
14 referências
public double TempoVoo
{
    get { return tempoVoo; }
    set { tempoVoo = value; }
}

4 referências
public string ResolucaoRealParaOAngulo
{
    get { return resolucaoRealParaOAngulo; }
    set { resolucaoRealParaOAngulo = value; }
}

1 referência
public double VelocidadeInicialEmXProjétil
{
    get { return velocidadeInicialEmXProjétil; }
    set { velocidadeInicialEmXProjétil = value; }
}

4 referências
public double VelocidadeInicialEmYProjétil
{
    get { return velocidadeInicialEmYProjétil; }
    set { velocidadeInicialEmYProjétil = value; }
}

5 referências
public double PosicaoEmYProjétil
{
    get { return posicaoEmYProjétil; }
    set { posicaoEmYProjétil = value; }
}
}
```

- TELA BEM-VINDO

```
//Declarando bibliotecas que serão utilizadas ao decorrer do código
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;

//Declarando o local e namespace no qual o código está alocado
namespace PPL_Main.Telas
{
    //Cria a classe de execução da tela
    4 referências
    public partial class TelaBemVindo : Form
    {
        //Cria um instanciamento inicial que inicializa a tela/Visual do WindowsForms
        1 referência
        public TelaBemVindo()
        {
            InitializeComponent();
        }

        1 referência
        private void label1_Click(object sender, EventArgs e)
        {
        }

        //Cria método que realiza a abertura do script em python que verifica a instalação das bibliotecas
        //necessárias para geração do gráfico ao final da execução da simulação executada na tela "TelaSimulacao"
        1 referência
        private void AbreScriptPython()
        {
            //O CAMINHO DO SCRIPT EM PYTHON DEVE SER ALTERADO CONFORME A MÁQUINA/COMPUTADOR QUE ESTIVER EXECUTANDO O PROGRAMA
            Process.Start("python", @"D:\Heitor\FESA\PBL\PPL_Main\PythonScripts\VerificacaoBibliotecas.py");
        }
    }
}
```

```
//Define ações que serão executadas ao clique do botão "buttonComecar"
1 referência
private void buttonComecar_Click(object sender, EventArgs e)
{
    //Cria um novo objeto a partir da classe "TelaInicio" e chama esse objeto com o nome de variavel "telaInicio"
    TelaInicio telaInicio = new TelaInicio();

    //Realiza a abertura da TelaInicio chamando método nativo de exibição da tela
    telaInicio.Show();

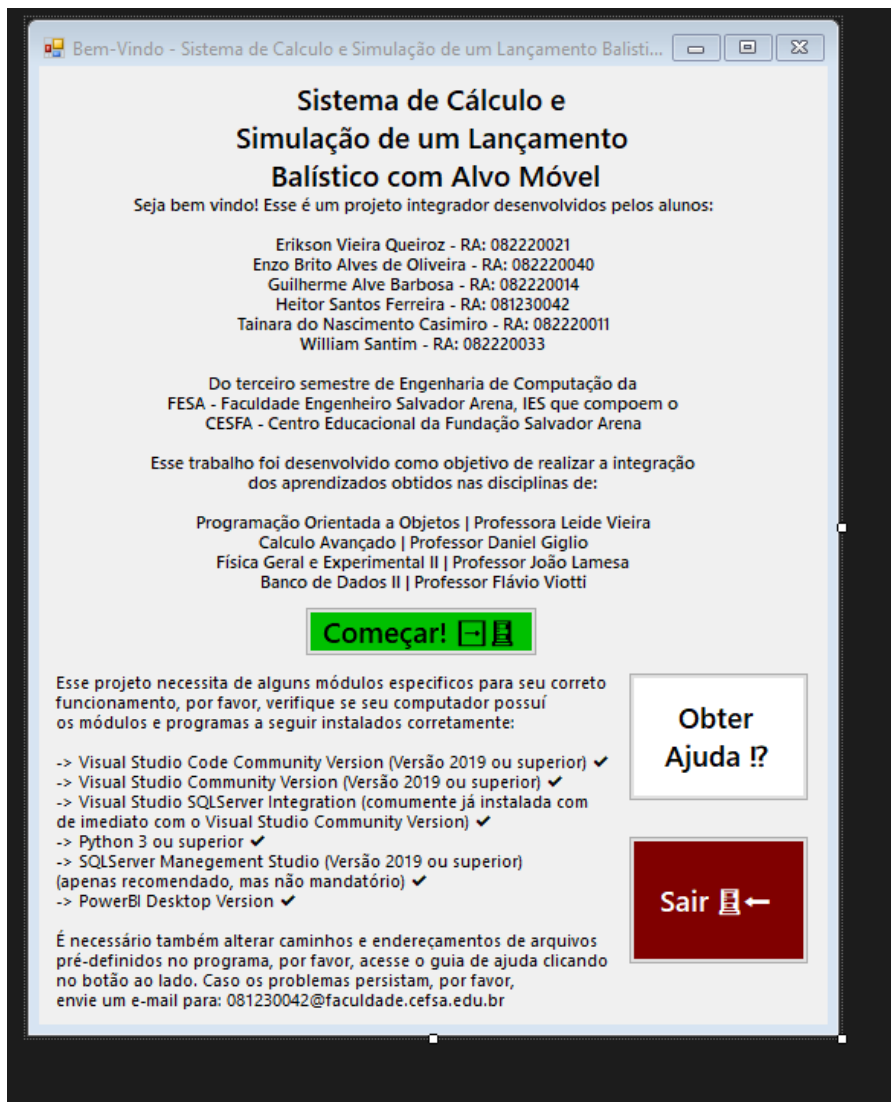
    //Realiza a execução do método "AbreScriptPython"
    AbreScriptPython();
}

//Define ações que serão executadas ao clique do botão "buttonObterAjuda"
1 referência
private void buttonObterAjuda_Click(object sender, EventArgs e) //O método de clique recebe dois parametros pré-definidos,
                                                                //o parametro de "sender" evoca um navegador local do computador no qual o software é executado
                                                                //e a classe de execuções de biblioteca "EventArgs" relaciona o navegador com o link definido dentro da função
{
    //Evoca uma variavel do tipo string que recebe o link no qual está contido o manual de adaptação do programa
    string link = "https://cefsaedu-my.sharepoint.com/:b:/g/personal/081230842_faculdade_cefsa_edu_br/Eevpcu5-LjH0rcRPo6S94U48u571y00oTE8pYMovVpG8FA?e=a9CvJ";

    //Executa a abertura do link por meio do processo de System.Diagnostics
    System.Diagnostics.Process.Start(link);
}

//Define ações que serão executadas ao clique do botão "buttonSair"
1 referência
private void buttonSair_Click(object sender, EventArgs e)
{
    //Realiza o encerramento da aplicação de uma maneira geral.
    Application.Exit();
}
}
```

- TELA BEM-VINDO (DESIGN)



```

4 referências
partial class TelaBemVindo
{
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
    0 referências
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    Windows Form Designer generated code

    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Button buttonComecar;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.Button buttonObterAjuda;
    private System.Windows.Forms.Button buttonSair;
}

```


- TELA CADASTRAR

```
//Declara o local e o namespace no qual o código estará alocado dentro do software
namespace PPL_Main.Telas
{
    //Cria a classe de execução da tela
    4 referências
    public partial class Telacadastrar : Form
    {
        //Realiza evocação das referências contidas no código principal de execução do programa
        Program referenciaClassePrograma = new Program();

        //Realiza instanciamento inicial do visual, inicializando componente do windowsForms
        1 referência
        public Telacadastrar()
        {
            InitializeComponent();
        }

        //Define um caracter especial que irá mascarar os valores digitados na textBox de senha
        1 referência
        private void senhaTextBox_TextChanged(object sender, EventArgs e)
        {
            textBoxCadastrarSenhaUsuario.PasswordChar = '*';
        }

        //Cria um método que realiza a criação ou alteração da SP (Stored Procedure) que irá cadastrar novos usuários
        1 referência
        internal void CriaSPCadastrarUsuario()
        {
            //Realiza o relacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
            //evocado anteriormente em 'referenciaClassePrograma'
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                //Realiza a abertura da conexão com o banco de dados
                conexaoBancoDeDados.Open();

                //Cria string que irá receber a SP (Stored Procedure)
                string criandoSPInclusaoDadosUsuario = @"

                --Cria Stored Procedure ou altera a mesma, definindo um nome para ela
                CREATE OR ALTER PROCEDURE InserirUsuario

                --Define o nome dos valores e a tipagem de dado que irão compor a SP
                @NomeUsuario VARCHAR(25),
                @SenhaUsuario VARCHAR(25)

                AS
```

```
BEGIN

    --Declara uma variável mutável e adaptativa que irá determinar um ID de usuário, o dado é definido como inteiro e recebe o próximo valor sequencial do último
    DECLARE @ProximoID INT;

    SELECT @ProximoID = ISNULL(MAX(id_usuario), 0) + 1 FROM usuarios;

    --Realiza a inserção dos novos dados obtidos na tabela já existente, declarando os valores que devem ser preenchidos e os campos que receberão esses valores
    INSERT INTO usuarios (id_usuario, nome_usuario, senha_usuario)
    VALUES (@ProximoID, @NomeUsuario, @SenhaUsuario)

END";

    //Cria instancia SQL que realiza a execução do comando de criação ou alteração da SP (Stored Procedure) declarada
    using (SqlCommand comandoQuerySPInseredadosUsuario = new SqlCommand(criandoSPInclusaoDadosUsuario, conexaoBancoDeDados))
    {
        comandoQuerySPInseredadosUsuario.ExecuteNonQuery();
    }
}

//Cria método que categoriza e gerencia todo o cadastramento do usuário
1 referência
internal void CadastraUsuario()
{
    //Julga se os valores digitados nas textboxes de Nome de usuário e de Senha de usuário estão recebendo os valores corretos de tipo string
    if (!(!string.IsNullOrEmpty(textBoxCadastrarNomeUsuario.Text) &&
        (!string.IsNullOrEmpty(textBoxCadastrarSenhaUsuario.Text))))
    {
        //Abre laço de checagem por tentativa
        try
        {
            //Realiza o relacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
            //evocado anteriormente em 'referenciaClassePrograma'
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                //Realiza a abertura da conexão com o banco de dados
                conexaoBancoDeDados.Open();
```

```

//Abre instancia SQL onde cria um comando em SQL utilizando a SP (Stored Procedure) 'InserirUsuario' para que a mesma seja acionada
//Receba os valores que serão ingeridos durante a construção da SP (Stored Procedure)
using (SqlCommand comandoInserirDadosTabelaUsuarios = new SqlCommand("InserirUsuario", conexaoBancoDeDados))
{
    //Define que o comando aberto acima é um comando do tipo SP (Stored Procedure)
    comandoInserirDadosTabelaUsuarios.CommandType = CommandType.StoredProcedure;

    //Define quais valores devem ser atribuídos a SP que está associada ao comando
    comandoInserirDadosTabelaUsuarios.Parameters.AddWithValue("@NomeUsuario", this.textBoxCadastrarNomeUsuario.Text.ToUpper());
    //Define que o valor a ser considerado no parametro '@NomeUsuario' da SP deve ser o valor da textBox que recebe o nome de usuário,
    //o mesmo converte o valor para maiusculo absoluto

    comandoInserirDadosTabelaUsuarios.Parameters.AddWithValue("@SenhaUsuario", this.textBoxCadastrarSenhaUsuario.Text);
    //Define que o valor a ser considerado no parametro '@SenhaUsuario' da SP deve ser o valor da textBox que recebe a senha do usuário

    //Executa o comando com parametro de "NonQuery", pois o mesmo se trata da evocação de uma SP
    comandoInserirDadosTabelaUsuarios.ExecuteNonQuery();

    //Exibe uma mensagem ao usuário por meio de uma caixa de mensagem
    MessageBox.Show("Dados cadastrados com sucesso!");

    //Realiza a 'limpeza' dos valores em ambas as textBox's existentes no WindowsForm em questão
    textBoxCadastrarNomeUsuario.Text = "";
    textBoxCadastrarSenhaUsuario.Text = "";

    //...Encerra o laço de checagem por tentativa
}

}

//Inicia laço de expressão
//Caso haja um erro na tentativa, uma exceção é aberta e recebe o valor de erro gerado naturalmente pelo SQL
catch (SqlException ex)
{
    //Exibe uma mensagem ao usuário por meio de uma caixa de mensagem, relatando ao mesmo os valores dos erros
    MessageBox.Show($"Erro SQL ({ex.Number}): {ex.Message}");

    //Realiza a 'limpeza' dos valores em ambas as textBox's existentes no WindowsForm em questão
    textBoxCadastrarNomeUsuario.Text = "";
    textBoxCadastrarSenhaUsuario.Text = "";
}
}
}

```

```

//Caso os valores digitados nas textBox's não sejam atendidos nas instancias definidas no conceito if, cria uma alternativa de execução
else
{
    //Exibe uma mensagem ao usuário por meio de uma caixa de mensagem
    MessageBox.Show("Os tipos de dados digitados estão incorretos, por favor, digite novamente.");

    //Realiza a 'limpeza' dos valores em ambas as textBox's existentes no WindowsForm em questão
    textBoxCadastrarNomeUsuario.Text = "";
    textBoxCadastrarSenhaUsuario.Text = "";
}

}

//Criando método que realiza a criação ou alteração do Trigger que verifica condições específicas de preenchimento do nome de usuário
//referencia
public void CriarTriggerVerificaNomeUsuario()
{
    //Realiza o relacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
    //evocado anteriormente em 'ReferenciaClassePrograma'
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
    {
        //Realiza a abertura da conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Criando comando de trigger
        string scriptTriggerVerificaNomeUsuario = @"
--Define criação ou alteração do trigger juntamente com o seu nome e sua determinação de alteração ou circunstância de ativação. Esse trigger é alterado após um INSERT INTO ou um UPDATE
CREATE OR ALTER TRIGGER VerificaNomeUsuario
ON usuarios
AFTER INSERT, UPDATE
AS
BEGIN
    --Declarando uma variável de mensagem de erro que irá receber o número máximo de caracteres permitido no SQLServer Local Statement
    SET NOCOUNT ON;

    DECLARE @ErrorMessage NVARCHAR(MAX);

    --Realizando a verificação da condição para gerar um erro. No caso, não é permitido que o usuário insira um nome de usuário que contenha números, caracteres especiais ou 'ç'
    --A verificação das condições é realizada após uma tentativa de inserção de novos dados na tabela
    IF EXISTS (
        SELECT 1
        FROM Inserted
        WHERE PATINDEX('%[a-zA-Z ]%', nome_usuario) > 0 OR
        nome_usuario LIKE '%[0-9]%' OR
        nome_usuario LIKE '%[ç]%'
    )
    BEGIN
        --Caso a verificação retorne um valor verdadeiro, no caso, caso o usuário digite um nome de usuário que contenha números, caracteres especiais ou 'ç',
        --a variável de mensagem de erro declarada anteriormente recebe um valor numérico e genérico de gravação em sistema (50000 no caso) e recebe um valor de mensagem para ser retornado
        SET @ErrorMessage = 'O nome de usuário não atende aos requisitos solicitados';
        THROW 50000, @ErrorMessage, 1;
        ROLLBACK;
    END
END;

";

    //Cria instancia SQL que realiza a execução do comando de criação ou alteração do Trigger declarado
    using (SqlCommand comandoCriarTriggerVerificaNomeUsuario = new SqlCommand(scriptTriggerVerificaNomeUsuario, conexaoBancoDeDados))
    {
        comandoCriarTriggerVerificaNomeUsuario.ExecuteNonQuery();
    }
}
}

```



```

1 referência
private void label1_Click(object sender, EventArgs e)
{
}

1 referência
private void TelaCadastrar_Load(object sender, EventArgs e)
{
}

//Define ações que serão realizadas após o clique do botão 'voltarButton'
1 referência
private void voltarButton_Click(object sender, EventArgs e)
{
    //Fecha a tela do WindowsForms atual
    this.Close();
}
}

```

- TELA CADASTRAR (DESIGN)

```

namespace PPL_Main.Telas
{
    4 referências
    partial class TelaCadastrar
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        0 referências
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows Form Designer generated code

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox textBoxCadastrSenhaUsuario;
        private System.Windows.Forms.TextBox textBoxCadastrNomeUsuario;
        private System.Windows.Forms.Button cadastrarButton;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Button voltarButton;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label5;
    }
}

```

- TELA ENTRADA

```
//Decalorando o local e namespace no qual o código está alocado
Namespace PPL_Main.Telas
{
    //Cria a classe de execução da tela
    4 referências
    public partial class TelaEntrar : Form
    {
        //Realiza evocação das referencias contidas no código principal de execução do programa
        Program referenciaClassePrograma = new Program();

        //Cria duas variáveis de nível publico que serão usadas no decorrer do código
        public string armazenaIDUsuarioString; //Declara variavel do tipo 'string' que armazena o ID do usuário autenticado
        public int armazenaIDUsuarioInt; //Declara variavel do tipo 'int' que armazena o ID do usuário autenticado

        //Cria um instanciamento inicial que inicializa a tela/Visual do WindowsForms
        1 referência
        public TelaEntrar()
        {
            InitializeComponent();
        }

        //Cria um método que realiza a criação ou alteração da SP (Stored Procedure) que irá verificar a existência do nome e senha de usuario digitados nas textBox's presentes no WindowsForms atual
        1 referência
        internal void CriaSPVerificaUsuario()
        {
            //Realiza o reacionamento e referencia da conexão com o banco de dados com o valor contido na variavel 'enderecamentoBancoDeDados' no programa de execução principal do software,
            //evocado anteriormente em 'referenciaClassePrograma'
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                //Realiza abertura da conexão com o banco de dados
                conexaoBancoDeDados.Open();

                //Cria SP (Stored Procedure)
                string criandoSPVerificacaoUsuario = @"
                --Cria Stored Procedure ou altera a mesma, definindo um nome para ela
                CREATE OR ALTER PROCEDURE VerificaUsuario

                --Define o nome dos valores e a tipagem de dado que irão compor a SP
                @NomeUsuario VARCHAR(25),
                @SenhaUsuario VARCHAR(25)

                AS
                BEGIN
```

```
--Declarando variavel de contagem que recebe uma tipagem de dado de número inteiro
DECLARE @Contagem INT;

--É realizada a contagem de usuários com o mesmo nome de usuario e senha digitada nas textBox's refereridas no visual do WindowsForms
SELECT @Contagem = COUNT(*)
FROM usuarios
WHERE nome_usuario = @NomeUsuario
AND senha_usuario = @SenhaUsuario;

--Caso a variavel de contagem seja maior que zero, significa que existe um cadastro para os valores de nome de usuario e senha digitados nas textBox's referidas no visual do WindowsForms
IF @Contagem > 0
BEGIN
    --Retorna o nome de usuário encontrado com os valores fornecidos
    SELECT nome_usuario
    FROM usuarios
    WHERE nome_usuario = @NomeUsuario;

END
ELSE
BEGIN
    --Caso não exista nenhum usuario cadastrado com os valores de nome de usuario e senha fornecidos, retorna um 'N/A' como nome de usuario. Esse valor será transformado ao decorrer do código
    SELECT 'N/A' AS nome_usuario

END

END";

//Cria instancia SQL que realiza a execução do comando de criação ou alteração da SP (Stored Procedure) declarada
using (SqlCommand comandoQuerySPVerificadosUsuario = new SqlCommand(criandoSPVerificacaoUsuario, conexaoBancoDeDados))
{
    comandoQuerySPVerificadosUsuario.ExecuteNonQuery();
}
}
```

```

//Cria um método que realiza a criação ou alteração da SP (Stored Procedure) que irá verificar a existência do nome e senha de usuário digitados nas textBox's presentes no WindowsForms atual,
//capturando somente o ID do usuário
//Referencia
internal void criaSPCapturaIDUsuarioAutenticado()
{
    //Realiza o reacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
    //evocado anteriormente em 'referenciaClassePrograma'
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
    {
        //Realiza abertura da conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Cria SP (Stored Procedure)
        string criandoSPCapturaIDUsuario = @"

--Cria Stored Procedure ou altera a mesma, definindo um nome para ela
CREATE OR ALTER PROCEDURE CapturaIDUsuarioAutenticado

--Define o nome dos valores e a tipagem de dado que irão compor a SP
    @nomeUsuario VARCHAR(25),
    @senhaUsuario VARCHAR(25)

AS

BEGIN

--Declarando variável de contagem que recebe uma tipagem de dado de número inteiro
DECLARE @Contagem INT;

--É realizada a contagem de usuários com o mesmo nome de usuário e senha digitada nas textBox's referenciadas no visual do WindowsForms
SELECT @Contagem = COUNT(*)
FROM usuarios
WHERE nome_usuario = @nomeUsuario
AND senha_usuario = @senhaUsuario;

--Caso a variável de contagem seja maior que zero, significa que existe um cadastro para os valores de nome de usuário e senha digitados nas textBox's referidas no visual do WindowsForms
IF @Contagem > 0
BEGIN
    --Retorna o ID de usuário encontrado com os valores fornecidos
    SELECT id_usuario
    FROM usuarios
    WHERE nome_usuario = @nomeUsuario;

END
ELSE
BEGIN

```

```

--Caso não exista nenhum usuário cadastrado com os valores de nome de usuário e senha fornecidos, retorna um '0' como ID de usuário. Esse valor será transformado ao decorrer do código
SELECT '0' AS id_usuario

END

END";

//Cria instância SQL que realiza a execução do comando de criação ou alteração da SP (Stored Procedure) declarada
using (SqlCommand comandoQuerySPCapturaIDUsuario = new SqlCommand(criandoSPCapturaIDUsuario, conexaoBancoDeDados))
{
    comandoQuerySPCapturaIDUsuario.ExecuteNonQuery();
}

}

//Criando método que realiza a autenticação do usuário
//Referencia
public void AutenticarUsuario()
{
    //Realiza o reacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
    //evocado anteriormente em 'referenciaClassePrograma'
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
    {
        //Realiza a abertura da conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Realiza a criação de um instanciamento de comando de SP (Stored Procedure), utilizando a SP 'VerificaUsuario'
        using (SqlCommand comandoAutenticarUsuario = new SqlCommand("VerificaUsuario", conexaoBancoDeDados))
        {
            //Define o comando como um evocador de Stored Procedure
            comandoAutenticarUsuario.CommandType = CommandType.StoredProcedure;

            //Atribui os valores que irão preencher os parâmetros declarados na SP
            comandoAutenticarUsuario.Parameters.AddWithValue("@nomeUsuario", textBoxEntrarNomeUsuario.Text.ToUpper());
            //Declara que o parâmetro da SP de "@nomeUsuario" irá receber o valor contido na textBox que armazena o nome de usuário,
            //convertendo o valor digitado para todas as letras sendo em maiúscula

            comandoAutenticarUsuario.Parameters.AddWithValue("@senhaUsuario", textBoxEntrarSenhaUsuario.Text);
            //Declara que o parâmetro da SP irá de "@senhaUsuario" irá receber o valor contido na textBox que armazena a senha de usuário

            //Declara uma variável do tipo 'var', ou seja, uma variável escaneadora de sistema que irá executar o comando de atribuição de valores a SP em questão.
            //A variável irá armazenar o valor retornado pela SP
            var resultadoAutenticacao = comandoAutenticarUsuario.ExecuteScalar();

```

```

//Realizando verificação. Caso o valor da variável de 'resultadoAutenticacao' seja diferente de vazio e seja diferente do valor 'N/A',
//irá executar as linhas de comando declaradas dentro do laço de verificação
if ((resultadoAutenticacao != null) &&
    (resultadoAutenticacao.ToString() != "N/A"))
{
    //Realiza a criação de um instanciamento de comando de SP (Stored Procedure), utilizando a SP 'CapturaIDUsuarioAutenticado'
    using (SqlCommand comandoCapturaIDUsuario = new SqlCommand("CapturaIDUsuarioAutenticado", conexaoBancoDeDados))
    {
        //Define o comando como um evocador de Stored Procedure
        comandoCapturaIDUsuario.CommandType = CommandType.StoredProcedure;

        //Atribui os valores que irão preencher os parametros declarados na SP
        comandoCapturaIDUsuario.Parameters.AddWithValue("@NomeUsuario", textBoxEntrarNomeUsuario.Text.ToUpper());
        //Declara que o parametro da SP de "@NomeUsuario" irá receber o valor contido na textBox que armazena o nome de usuario,
        //convertendo o valor digitado para todas as letras sendo em maiuscula
        comandoCapturaIDUsuario.Parameters.AddWithValue("@SenhaUsuario", textBoxEntrarSenhaUsuario.Text);
        //Declara que o parametro da SP irá de "@SenhaUsuario" irá receber o valor contido na textBox que armazena a senha de usuario

        //Declara uma variável do tipo 'var', ou seja, uma variável escaneadora de sistema que irá executar o comando de atribuição de valores a SP em questão.
        //A variável irá armazenar o valor retornado pela SP
        var resultadoID = comandoCapturaIDUsuario.ExecuteScalar();

        //Realiza conversões de tipagem de dado e formatação do valor contido na variável 'resultadoID'
        armazenarIDUsuarioString = resultadoID.ToString();
        //Converte o valor da variável para 'string' em modelo de pura atribuição e atribui esse valor a variável do tipo 'string' com o nome de 'armazenarIDUsuarioString'

        armazenarIDUsuarioInt = int.Parse(armazenarIDUsuarioString);
        //Converte o valor da variável 'armazenarIDUsuarioString' em 'int' e atribui esse valor a variável 'armazenarIDUsuarioInt'
    }

    //Caso todas as verificações sejam realizadas com valor de verdade (True) e sejam executadas com sucesso,
    //retorna uma caixa de mensagem dando boas vindas ao usuario autenticado, juntamente com seu ID de usuario
    MessageBox.Show("Seja bem-vindo " + resultadoAutenticacao.ToString() + "\n" +
        "Seu ID é: " + armazenarIDUsuarioInt);

    //Evoca a tela atual que está em execução e chama a mesma por uma variável de 'telaAtual'
    Form telaAtual = Form.ActiveForm;

    //Evoca um novo objeto de tela de menu e atribui com o nome de 'telaMenu'
    TelaMenu telaMenu = new TelaMenu();

    //Abre a tela de menu
    telaMenu.Show();

    //Fecha a tela atual
    telaAtual.Close();
}
}

```

```

    }
    else
    {
        MessageBox.Show("Nome de usuário ou senha incorretos");
    }
}

//Define as ações que serão executadas ao clique do botão 'buttonVoltar'
1 referência
private void buttonVoltar_Click(object sender, EventArgs e)
{
    //Fecha a tela atual
    this.Close();
}

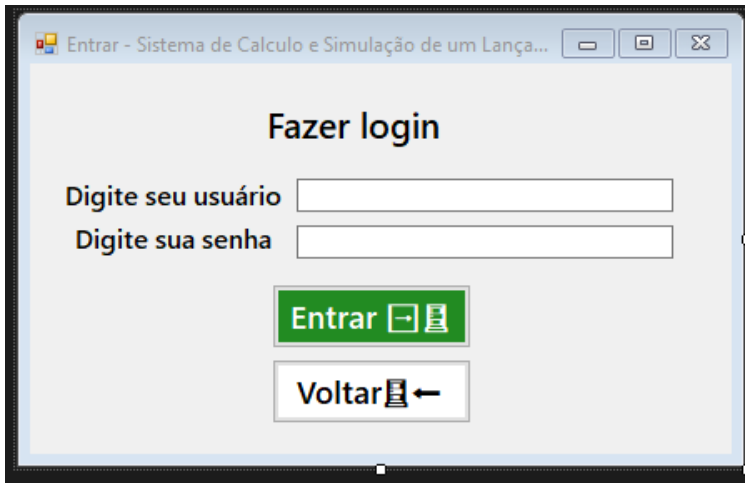
//Define as ações que serão executadas ao clique do botão 'buttonEntrar'
1 referência
private void buttonEntrar_Click(object sender, EventArgs e)
{
    //Realiza a execução dos métodos que tem relação com SP's
    criaSPCapturaIDUsuarioAutenticado(); //Executa método que cria ou altera a SP que realiza a captura do ID do usuário
    criaSPVerificaUsuario(); // Executa o método que cria ou altera a SP que realiza a verificação do usuario

    //Executa o método que realiza a autenticação do usuario
    AutenticarUsuario();
}

//Define um caracter especial que irá mascarar os valores digitados na textBox de senha
1 referência
private void textBoxEntrarSenhaUsuario_TextChanged(object sender, EventArgs e)
{
    textBoxEntrarSenhaUsuario.PasswordChar = '*';
}
}
}

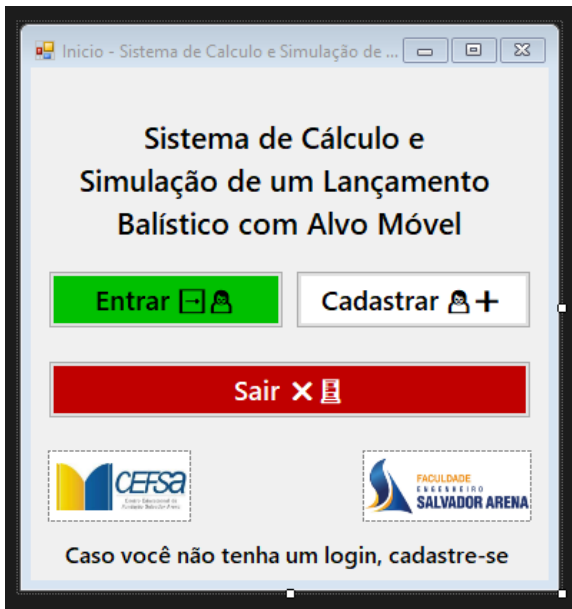
```

- TELA ENTRADA (DESIGN)



The image shows a login window titled "Entrar - Sistema de Cálculo e Simulação de um Lança...". The window has a light gray background and a white border. At the top, the title bar shows the window title and standard Windows controls (minimize, maximize, close). The main content area is titled "Fazer login" in bold black text. Below the title, there are two input fields: "Digite seu usuário" and "Digite sua senha". Below the input fields, there are two buttons: "Entrar" (green button with a white right-pointing arrow icon) and "Voltar" (white button with a black left-pointing arrow icon).

- TELA DE INÍCIO (DESIGN)



The image shows a home window titled "Inicio - Sistema de Cálculo e Simulação de ...". The window has a light gray background and a white border. At the top, the title bar shows the window title and standard Windows controls (minimize, maximize, close). The main content area is titled "Sistema de Cálculo e Simulação de um Lançamento Balístico com Alvo Móvel" in bold black text. Below the title, there are three buttons: "Entrar" (green button with a white right-pointing arrow icon), "Cadastrar" (white button with a black right-pointing arrow icon), and "Sair" (red button with a white right-pointing arrow icon). Below the buttons, there are two logos: "CEPSA" and "FACULDADE ENGENHEIRO SALVADOR ARENA". At the bottom, there is a text message: "Caso você não tenha um login, cadastre-se".


```

//Declarando nome da "namespace"/"ambiente" no qual o programa se encontra
namespace PPL_Main.Telas
{
    //Declarando a construção da tela inicial
    7 referências
    public partial class TelaInicio : Form
    {
        //Instanciando construtor da TelaEntrar
        2 referências
        public TelaInicio()
        {
            InitializeComponent();
        }

        //Atribuindo ações para o botão de Entrar
        1 referência
        private void EntrarButton_Click(object sender, EventArgs e)
        {
            //Evoca um novo objeto da 'TelaEntrar' e atribui com o nome de 'telaEntrar'
            TelaEntrar telaEntrar = new TelaEntrar();

            //Exibe a TelaEntrar
            telaEntrar.Show();
        }

        //Atribuindo ações para o botão de "Sair"
        1 referência
        private void SairButton_Click(object sender, EventArgs e)
        {
            //Fechando a aplicação completamente
            Application.Exit();
        }

        //Definindo ações para o botão de "Cadastrar"
        1 referência
        private void cadastrarButton_Click(object sender, EventArgs e)
        {
            //Evoca um novo objeto da 'TelaCadastrar' e atribui com o nome de 'telaCadastrar'
            TelaCadastrar telaCadastrar = new TelaCadastrar();

            //Exibe a TelaCadastrar
            telaCadastrar.Show();
        }
    }
}

```

- TELA MENU

```
//Decalando o local e namespace no qual o código está alocado
namespace PPL_Main.Telas
{
    //Declarando instanciamento da classe do visual
    public partial class TelaMenu : Form
    {
        //Importando um novo instanciamento que referencia o código principal do programa, para que esse trecho do código possa ter acesso as variáveis contidas dentro desse trecho
        Program referenciaClassePrograma = new Program();

        //Inicializando visual do WindowsForms
        public TelaMenu()
        {
            InitializeComponent();
        }

        //Criando método que seleciona todo o conteúdo da tabela de simulações e realiza o salvamento do conteudo em um arquivo '.csv'
        private void SeleccionaConteudosSimulacoes()
        {
            //Define comando que seleciona todo o conteúdo da tabela de simulações
            string querySelecionaTudoTabelaSimulacoes = "SELECT * FROM simulacoes";

            //Definindo o caminho onde o arquivo '.csv' será salvo
            //ESSE CAMINHO DEVE SER ALTERADO CONFORME A NECESSIDADE DO EXECUTOR DO PROGRAMA, ALTERANDO PARA OS PADRÕES PARA SE ADEQUAR A MÁQUINA NO QUAL O SOFTWARE ESTÁ SENDO EXECUTADO
            string caminhoArquivo = @"D:\Heitor\FESA\PBL\PPL_Main\RelatorioPaginado\Dados\simulacoes.csv";

            //Criando laço de tentativa de execução
            try
            {
                // Conecta ao servidor SQL
                using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
                {
                    conexaoBancoDeDados.Open();

                    // Cria um adaptador SQL para executar a consulta
                    using (SqlDataAdapter adapter = new SqlDataAdapter(querySelecionaTudoTabelaSimulacoes, conexaoBancoDeDados))
                    {
                        // Cria um DataSet para armazenar os resultados da consulta
                        DataSet dataSet = new DataSet();

                        // Preenche o DataSet com os resultados da consulta
                        adapter.Fill(dataSet);

                        // Chama o método para salvar o DataSet em um arquivo CSV, passando o caminho do arquivo
                        SaveDataSetToCsv(dataSet, caminhoArquivo);
                    }
                }
            }
            //...Encerra laço de tentativa
        }
    }
}
```

```
//Criando laço de exceção
catch (Exception ex)
{
    //Exibe caixa de mensagem que denuncia para o usuário o erro de processamento que ocorreu
    MessageBox.Show("Erro: " + ex.Message);
}

//Criando método que seleciona todo o conteúdo da tabela de projétil e realiza o salvamento do conteudo em um arquivo '.csv'
private void SeleccionaConteudoProjatil()
{
    //Define comando que seleciona todo o conteúdo da tabela de simulações
    string querySelecionaTudoTabelaProjatil = "SELECT * FROM projatil";

    //Definindo o caminho onde o arquivo '.csv' será salvo
    //ESSE CAMINHO DEVE SER ALTERADO CONFORME A NECESSIDADE DO EXECUTOR DO PROGRAMA, ALTERANDO PARA OS PADRÕES PARA SE ADEQUAR A MÁQUINA NO QUAL O SOFTWARE ESTÁ SENDO EXECUTADO
    string caminhoArquivo = @"D:\Heitor\FESA\PBL\PPL_Main\RelatorioPaginado\Dados\projatil.csv";

    //Cria laço de tentativa de execução
    try
    {
        // Conecta ao servidor SQL
        using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
        {
            conexaoBancoDeDados.Open();

            // Cria um adaptador SQL para executar a consulta
            using (SqlDataAdapter adapter = new SqlDataAdapter(querySelecionaTudoTabelaProjatil, conexaoBancoDeDados))
            {
                // Cria um DataSet para armazenar os resultados da consulta
                DataSet dataSet = new DataSet();

                // Preenche o DataSet com os resultados da consulta
                adapter.Fill(dataSet);

                // Chama o método para salvar o DataSet em um arquivo CSV, passando o caminho do arquivo
                SaveDataSetToCsv(dataSet, caminhoArquivo);
            }
        }
    }
    //...Encerra laço de tentativa
}
```

```

        //Criando laço de exceção
        catch (Exception ex)
        {
            //Exibe caixa de mensagem que denuncia para o usuário o erro de processamento que ocorreu
            MessageBox.Show("Erro: " + ex.Message);
        }
    }

    //Criando método que seleciona todo o conteúdo da tabela de alvo e realiza o salvamento do conteúdo em um arquivo '.csv'
    1 referência
    private void SelecionaConteudoAlvo()
    {
        //Define comando que seleciona todo o conteúdo da tabela de simulações
        string querySelecionaTudoTabelaAlvo = "SELECT * FROM alvo";

        //Definindo o caminho onde o arquivo '.csv' será salvo
        //ESSE CAMINHO DEVE SER ALTERADO CONFORME A NECESSIDADE DO EXECUTOR DO PROGRAMA, ALTERANDO PARA OS PADRÕES PARA SE ADEQUAR A MÁQUINA NO QUAL O SOFTWARE ESTÁ SENDO EXECUTADO
        string caminhoArquivo = @"D:\Heitor\FESA\PBL\PP\Main\RelatorioPaginado\Dados\alvo.csv";

        //Cria laço de tentativa de execução
        try
        {
            // Conecta ao servidor SQL
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                conexaoBancoDeDados.Open();

                // Cria um adaptador SQL para executar a consulta
                using (SqlDataAdapter adapter = new SqlDataAdapter(querySelecionaTudoTabelaAlvo, conexaoBancoDeDados))
                {
                    // Cria um DataSet para armazenar os resultados da consulta
                    DataSet dataset = new DataSet();

                    // Preenche o DataSet com os resultados da consulta
                    adapter.Fill(dataset);

                    // Chama o método para salvar o DataSet em um arquivo CSV, passando o caminho do arquivo
                    SaveDataSetToCSV(dataset, caminhoArquivo);
                }
            }

            //...Encerra laço de tentativa de execução
        }
    }

```

```

        //Cria laço de exceção
        catch (Exception ex)
        {
            //Exibe caixa de mensagem que denuncia para o usuário o erro de processamento que ocorreu
            MessageBox.Show("Erro: " + ex.Message);
        }
    }

    //Criando método que seleciona apenas o ID do usuário da tabela de usuarios e realiza o salvamento do conteúdo em um arquivo '.csv'
    1 referência
    private void SelecionaConteudoUsuario()
    {
        //Define comando que seleciona todo o conteúdo da tabela de simulações
        string querySelecionaTudoTabelaUsuario = "SELECT id_usuario FROM usuarios";

        //Definindo o caminho onde o arquivo '.csv' será salvo
        //ESSE CAMINHO DEVE SER ALTERADO CONFORME A NECESSIDADE DO EXECUTOR DO PROGRAMA, ALTERANDO PARA OS PADRÕES PARA SE ADEQUAR A MÁQUINA NO QUAL O SOFTWARE ESTÁ SENDO EXECUTADO
        string caminhoArquivo = @"D:\Heitor\FESA\PBL\PP\Main\RelatorioPaginado\Dados\usuarios.csv";

        //Cria laço de tentativa de execução
        try
        {
            // Conecta ao servidor SQL
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                conexaoBancoDeDados.Open();

                // Cria um adaptador SQL para executar a consulta
                using (SqlDataAdapter adapter = new SqlDataAdapter(querySelecionaTudoTabelaUsuario, conexaoBancoDeDados))
                {
                    // Cria um DataSet para armazenar os resultados da consulta
                    DataSet dataset = new DataSet();

                    // Preenche o DataSet com os resultados da consulta
                    adapter.Fill(dataset);

                    // Chama o método para salvar o DataSet em um arquivo CSV, passando o caminho do arquivo
                    SaveDataSetToCSV(dataset, caminhoArquivo);
                }
            }

            //...Encerra o laço de tentativa de execução
        }
    }

```

```

        //Cria laço de exceção
        catch (Exception ex)
        {
            //Exibe caixa de mensagem que denuncia para o usuário o erro de processamento que ocorreu
            MessageBox.Show("Erro: " + ex.Message);
        }
    }

    //Cria método que irá realizar o salvamento dos dados nos arquivos '.csv'
    4 referências
    static void SaveDataSetToCSV(DataSet dataSet, string filePath)
    {
        //Inicia método de StreamWriter de arquivos, inserindo o caminho do arquivo como referência
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            //Cria laço de checagem e repetição, onde para cada tabela contida no 'DataSet' criado, ele importa todo o conteúdo
            foreach (DataTable table in dataSet.Tables)
            {
                // Escreve o cabeçalho (nomes das colunas)
                foreach (DataColumn col in table.Columns)
                {
                    writer.Write($"{col.ColumnName};");
                }
                writer.WriteLine();

                // Escreve os dados
                foreach (DataRow row in table.Rows)
                {
                    foreach (var field in row.ItemArray)
                    {
                        writer.Write($"{field.ToString()};");
                    }
                    writer.WriteLine();
                }
            }
        }
    }
}

```

```

    //Cria método que realiza a abertura do arquivo do PowerBI do relatório páginado
    1 referência
    static void AbrirArquivoPBIX(string caminhoArquivoPBIX) //Esse método é de meio estático e recebe o caminho do arquivo do PowerBI Desktop
    {
        //Cria laço de tentativa de execução
        try
        {
            //Inicia processamento do caminho do arquivo
            Process.Start(caminhoArquivoPBIX);

            //...Encerra a tentativa de execução
        }

        //Cria laço de exceção
        catch (Exception ex)
        {
            //Exibe caixa de mensagem com o erro ocorrido
            MessageBox.Show($"Erro ao abrir o arquivo PBIX: {ex.Message}");
        }
    }

    1 referência
    private void TelaPrincipal_Load(object sender, EventArgs e)
    {
        //Define ações que serão executadas após o clique do botão 'buttonAbreTelaSimular'
        1 referência
        private void buttonAbreTelaSimular_Click(object sender, EventArgs e)
        {
            //Define um novo objeto que referencia o visual do WindowsForms atual
            Form telaAtual = Form.ActiveForm;

            //Inicia a criação de um novo objeto que referencia a TelaSimulacao com o nome de 'telaSimulacao'
            TelaSimulacao telaSimulacao = new TelaSimulacao();

            //Abre a tela de simulacao
            telaSimulacao.Show();

            //Fecha a tela atual
            telaAtual.Close();
        }
    }
}

```

```
//Define ações que serão executadas após o clique do botão 'buttonExibirRelatorioRepaginado'
1 referencia
private void buttonExibirRelatorioRepaginado_Click(object sender, EventArgs e)
{
    //Executa todos os métodos que execução de seleção de conteúdo das tabelas
    SeleccionaConteudosSimulacoes();
    SeleccionaConteudoAlvo();
    SeleccionaConteudoProjatil();
    SeleccionaConteudoUsuario();

    //Cria uma string que contém o caminho do arquivo do PowerBI
    //ESSE CAMINHO DEVE SER ALTERADO CONFORME A NECESSIDADE DO USUARIO E DA MÁQUINA NA QUAL O SOFTWARE ESTÁ SENDO EXECUTADA
    string caminhoRelatorioPaginado = @"D:\Heitor\FESA\PBL\PPL_Main\RelatorioPaginado\PBL_-_Relatorio_Paginado.pbix";

    //Executa o método que abre o arquivo PowerBI, referenciando o caminho do arquivo do PowerBI
    AbrirArquivoPBIX(caminhoRelatorioPaginado);
}

//Define ações que serão executadas após o clique do botão 'buttonConferirResultadosAnteriores'
1 referencia
private void buttonConferirResultadosAnteriores_Click(object sender, EventArgs e)
{
    //Define um novo objeto que referencia o visual do WindowsForms atual
    Form telaAtual = Form.ActiveForm;

    //Inicia a criação de um novo objeto que referencia a TelaResultadosAnteriores com o nome de 'telaResultadosAnteriores'
    TelaResultadosAnteriores telaResultadosAnteriores = new TelaResultadosAnteriores();

    //Abre tela de Resultados Anteriores
    telaResultadosAnteriores.Show();

    //Fecha a tela atual
    telaAtual.Close();
}
}
```

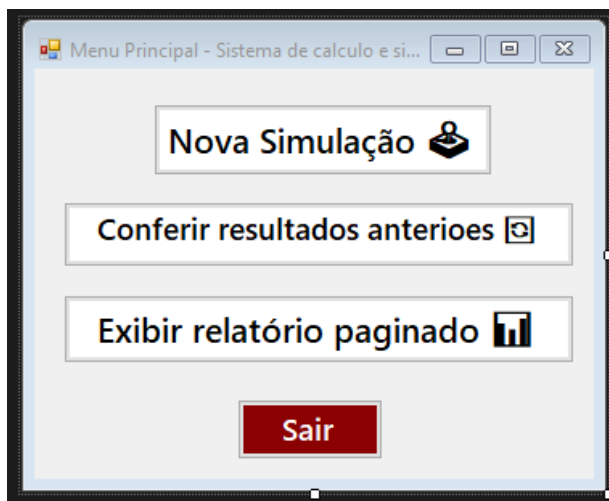
```
//Define ações que serão executadas após o clique do botão 'sairButton'
1 referencia
private void sairButton_Click(object sender, EventArgs e)
{
    //Define um novo objeto que referencia o visual do WindowsForms atual
    Form telaAtual = Form.ActiveForm;

    //Inicia a criação de novo objeto que referencia a TelaInicio com o nome de 'telaInicio'
    TelaInicio telaInicio = new TelaInicio();

    //Abre tela de Inicio
    telaInicio.Show();

    //Fecha tela atual
    telaAtual.Close();
}
}
```

- TELA MENU (DESIGN)



- TELA RESULTADOS ANTERIORES

```
//Define o local e o namespace no qual o arquivo está localizado
namespace PPL_Main.Telas
{
    //Define a classe do visual como instanciamento público
    4 referências
    public partial class TelaResultadosAnteriores : Form
    {
        //Referencia o código do programa principal, para que os métodos e variáveis declaradas possam ser acessíveis nesse trecho do código
        Program referenciaClassePrograma = new Program();

        //Inicializa o visual do WindowsForms juntamente com a execução do método e exibição e obtenção dos dados
        1 referência
        public TelaResultadosAnteriores()
        {
            InitializeComponent();
            ObterDados();
        }

        //Realiza a construção do método que obtém os dados das principais tabelas do programa e preenche os DataGridView's
        1 referência
        private void ObterDados()
        {
            //Realiza o relacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
            //evocado anteriormente em 'referenciaClassePrograma'
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                //Abre conexão com o banco de dados
                conexaoBancoDeDados.Open();

                //Cria comando que seleciona todo o conteúdo da tabela de simulações
                var sqlQueryTabelasSimulacoes = "SELECT * FROM simulacoes";

                //Utiliza adaptador SQL referenciando o comando e a conexão com o banco de dados
                using (SqlDataAdapter dasimulacoes = new SqlDataAdapter(sqlQueryTabelasSimulacoes, conexaoBancoDeDados))
                {
                    //Preenche o dataGridView de simulações com o conteúdo coletado na 'query'
                    using (DataTable dtsimulacoes = new DataTable())
                    {
                        dasimulacoes.Fill(dtsimulacoes);
                        dataGridViewTabelasSimulacoes.DataSource = dtsimulacoes;
                    }
                }
            }
        }
    }
}
```

```
//Define o local e o namespace no qual o arquivo está localizado
namespace PPL_Main.Telas
{
    //Define a classe do visual como instanciamento público
    4 referências
    public partial class TelaResultadosAnteriores : Form
    {
        //Referencia o código do programa principal, para que os métodos e variáveis declaradas possam ser acessíveis nesse trecho do código
        Program referenciaClassePrograma = new Program();

        //Inicializa o visual do WindowsForms juntamente com a execução do método e exibição e obtenção dos dados
        1 referência
        public TelaResultadosAnteriores()
        {
            InitializeComponent();
            ObterDados();
        }

        //Realiza a construção do método que obtém os dados das principais tabelas do programa e preenche os DataGridView's
        1 referência
        private void ObterDados()
        {
            //Realiza o relacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
            //evocado anteriormente em 'referenciaClassePrograma'
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                //Abre conexão com o banco de dados
                conexaoBancoDeDados.Open();

                //Cria comando que seleciona todo o conteúdo da tabela de simulações
                var sqlQueryTabelasSimulacoes = "SELECT * FROM simulacoes";

                //Utiliza adaptador SQL referenciando o comando e a conexão com o banco de dados
                using (SqlDataAdapter dasimulacoes = new SqlDataAdapter(sqlQueryTabelasSimulacoes, conexaoBancoDeDados))
                {
                    //Preenche o dataGridView de simulações com o conteúdo coletado na 'query'
                    using (DataTable dtsimulacoes = new DataTable())
                    {
                        dasimulacoes.Fill(dtsimulacoes);
                        dataGridViewTabelasSimulacoes.DataSource = dtsimulacoes;
                    }
                }
            }
        }
    }
}
```

- TELA RESULTADOS ANTERIORES (DESIGN)



```

namespace PPL_Main.Telas
{
    4 referências
    partial class TelaResultadosAnteriores
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        0 referências
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows Form Designer generated code

        private System.Windows.Forms.DataGridView dataGridViewTabelaSimulacoes;
        private System.Windows.Forms.Label labelDadosParaAlvo;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.DataGridView dataGridViewTabelaProjatil;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.DataGridView dataGridViewTabelaAlvo;
    }
}

```

• TELA SIMULAÇÃO

```
//Declarando local onde o código está alocado
namespace PPI_Main.Telas
{
    //Declarando a construção do visual
    4 referências
    public partial class TelaSimulacao : Form
    {
        //Declara um instanciamento de referencia ao código principal do programa, para que seja possível usar os métodos e variáveis declaradas ao decorrer do código
        Program referenciaClassePrograma = new Program();

        //Declarando instanciamento e criação de novo objeto de alvo
        CodigosMain.Alvo alvo1 = new CodigosMain.Alvo();

        //Declarando instanciamento e criação de novo objeto de projétil
        CodigosMain.Projetil projetil1 = new CodigosMain.Projetil();

        //Declarando instanciamento e criação de novo objeto de valores de calculo
        CodigosMain.ValoresCalculoEquacaoQuadratica valoresCalculo1 = new CodigosMain.ValoresCalculoEquacaoQuadratica();

        //Inicializando visual
        1 referência
        public TelaSimulacao()
        {
            InitializeComponent();
        }

        //Criando método que cria a Stored Procedure que realiza a criação do projétil
        1 referência
        internal void CriaSPCadastraProjelil()
        {
            //Realiza o reacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
            //evocado anteriormente em 'referenciaClassePrograma'
            using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
            {
                //Abrindo conexão com o banco de dados
                conexaoBancoDeDados.Open();

                //Criando comando de SP (Stored Procedure) que realiza a inclusão de novos dados na tabela de projétil
                string criandoSPInclusaoDadosProjelil = @"
                --Declara nome e estado de criação ou alteração da SP
                CREATE OR ALTER PROCEDURE InserirProjelil

                --Declara variáveis e tipagem de dados dos parametros da SP
                @AnguloTrajetoriaProjelil DECIMAL(10,2),
                @VelocidadeInicial1Projelil DECIMAL(10,2),
                @VelocidadeInicial2Projelil DECIMAL(10,2),
                @VelocidadeUsadaProjelil DECIMAL(10,2),
                @TempoVooProjelil DECIMAL(10,2),
                @RipsoMovimentoProjelil VARCHAR(30),
                @ExistenciaResolucaoRealAnguloFornecido VARCHAR(3)

                AS
```

```
BEGIN

    --Declara variável que irá receber um próximo ID de projétil, verificando sempre qual o ID máximo e encontrando o próximo número
    DECLARE @ProximoID INT;

    SELECT @ProximoID = ISNULL(MAX(id_projelil), 0) + 1 FROM projetil;

    --Declarando os campos que receberão inserção de dados e quais valores serão inseridos nos respectivos campos
    INSERT INTO projetil (id_projelil, angulo_trajetoria_projelil, velocidade_inicial1_projelil, velocidade_inicial2_projelil,
        velocidade_usada_projelil, tempo_de_voo_projelil, tipo_de_movimento_projelil, existencia_resolucao_real_angulo_fornecido)

    VALUES (@ProximoID, @AnguloTrajetoriaProjelil, @VelocidadeInicial1Projelil, @VelocidadeInicial2Projelil, @VelocidadeUsadaProjelil,
        @TempoVooProjelil, @RipsoMovimentoProjelil, @ExistenciaResolucaoRealAnguloFornecido);

END";

//Cria instancia SQL que realiza a execução do comando de criação ou alteração da SP (Stored Procedure) declarada
using (SqlCommand comandoQuerySPInseredadosProjelil = new SqlCommand(criandoSPInclusaoDadosProjelil, conexaoBancoDeDados))
{
    comandoQuerySPInseredadosProjelil.ExecuteNonQuery();
}
}

//Criando metodo que insere os dados do Projelil na tabela
2 referências
internal void CadastraProjelil()
{
    //Realiza o reacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
    //evocado anteriormente em 'referenciaClassePrograma'
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
    {
        //Abre conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Declara um comando em SQL que irá usar a SP 'InserirProjelil'
        using (SqlCommand comandoInseredadosTabelaProjelil = new SqlCommand("InserirProjelil", conexaoBancoDeDados))
        {
            //Declara o comando como uma SP
            comandoInseredadosTabelaProjelil.CommandType = CommandType.StoredProcedure;

            //Declara quais valores irão ser atribuídos em cada parametro da SP
            comandoInseredadosTabelaProjelil.Parameters.AddWithValue("@AnguloTrajetoriaProjelil", Convert.ToDecimal(projelil1.AnguloLancamentoProjelil));
            //Atribui o valor de angulo do projelil, convertido em decimal para o parametro "@AnguloTrajetoriaProjelil"

            comandoInseredadosTabelaProjelil.Parameters.AddWithValue("@VelocidadeInicial1Projelil", Convert.ToDecimal(projelil1.VelocidadeInicial1Projelil));
            //Atribui o valor de velocidade inicial 1 do projelil, convertido em decimal para o parametro "@VelocidadeInicial1Projelil"

            comandoInseredadosTabelaProjelil.Parameters.AddWithValue("@VelocidadeInicial2Projelil", Convert.ToDecimal(projelil1.VelocidadeInicial2Projelil));
            //Atribui o valor de velocidade inicial 2 do projelil, convertido em decimal para o parametro "@VelocidadeInicial2Projelil"

            comandoInseredadosTabelaProjelil.Parameters.AddWithValue("@VelocidadeUsadaProjelil", Convert.ToDecimal(projelil1.VelocidadeUsada));
            //Atribui o valor de velocidade usada do projelil, convertido em decimal para o parametro "@VelocidadeUsadaProjelil"
```



```

        comandoInseridosTabelaProjtil.Parameters.AddWithValue("@TempoVooProjtil", Convert.ToDecimal(projtil.TempoVoo));
        //Atribui o valor de voo do projtil, convertido em decimal para o parametro de "TempoVooProjtil"

        comandoInseridosTabelaProjtil.Parameters.AddWithValue("@TipoMovimentoProjtil", projtil.TipoMovimento);
        //Atribui o valor do tipo de movimento para o parametro de "TipoMovimentoProjtil"

        comandoInseridosTabelaProjtil.Parameters.AddWithValue("@ExistenciaResolucaoRealAnguloFornecido", projtil.ResolucaoRealParaAngulo);
        //Atribui o valor da existência real para ângulo fornecido para o parametro de "@ExistenciaResolucaoRealAnguloFornecido"

        //Executa o comando como um método de 'NonQuery', tendo em vista que o comando trata-se de uma SP
        comandoInseridosTabelaProjtil.ExecuteNonQuery();
    }
}

//Criando método que cria a SP que insere os dados do Alvo
1 referencia
internal void CriaSPCadastrarAlvo()
{
    //Realiza o reacionamento e referencia da conexão com o banco de dados com o valor contido na variável 'enderecamentoBancoDeDados' no programa de execução principal do software,
    //enviado anteriormente em 'ReferenciaClassePrograma'
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
    {
        //Abrindo conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Cria comando que cria ou altera SP
        string criandoSPInclusaoDadosAlvo = @"

        --Declara nome e estado de criação ou alteração da SP
        CREATE OR ALTER PROCEDURE InserirAlvo

        --Declara os parametros e suas tipagem de dados
        @AlturaInicialAlvo DECIMAL(10,2),
        @DistanciaCanhaoAlvo DECIMAL(10,2),
        @VelocidadeInicialAlvo DECIMAL(10,2),
        @AnguloMaximoAcertoAlvo DECIMAL(10,2),
        @TempoAteatingirSoloAlvo DECIMAL(10,2)

        AS

        BEGIN

```

```

        --Declara variável que irá receber um próximo ID de projtil, verificando sempre qual o ID máximo e encontrando o próximo número
        DECLARE @ProximoID INT;

        SELECT @ProximoID = ISNULL(MAX(id_alvo), 0) + 1 FROM alvo;

        --Declarando os campos que receberão inserção de dados e quais valores serão inseridos nos respectivos campos
        INSERT INTO alvo (id_alvo, altura_inicial_alvo, distancia_canhao_alvo, velocidade_inicial_alvo, angulo_maximo_acerto_alvo, tempo_ate_attingir_solo_alvo)
        VALUES (@ProximoID, @AlturaInicialAlvo, @DistanciaCanhaoAlvo, @VelocidadeInicialAlvo, @AnguloMaximoAcertoAlvo, @TempoAteatingirSoloAlvo);

    END";

    //Cria instancia SQL que realiza a execução do comando de criação ou alteração da SP
    using (SqlCommand comandoQuerySPInseridosAlvo = new SqlCommand(criandoSPInclusaoDadosAlvo, conexaoBancoDeDados))
    {
        comandoQuerySPInseridosAlvo.ExecuteNonQuery();
    }
}

//Criando método que realiza o cadastro do alvo
2 referencias
internal void CadastraAlvo()
{
    //Abre conexão com o banco de dados com o endereçamento referenciado no momento de instancia do programa
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
    {
        //Abre conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Inicia as variações de comando com a Stored Procedure de "InserirAlvo"
        using (SqlCommand comandoInseridosTabelaAlvo = new SqlCommand("InserirAlvo", conexaoBancoDeDados))
        {
            //Referencia o comando de inserir dados na tabela de alvo como uma SP
            comandoInseridosTabelaAlvo.CommandType = CommandType.StoredProcedure;

            //Atribui os valores de referencia na SP de "Inserir Alvo"
            comandoInseridosTabelaAlvo.Parameters.AddWithValue("@AlturaInicialAlvo", Convert.ToDecimal(alvo1.AlturaInicialAlvo));
            //Referencia o atributo da SP de "AlturaInicial" como o atributo de "AlturaInicialAlvo" do objeto Alvo

            comandoInseridosTabelaAlvo.Parameters.AddWithValue("@DistanciaCanhaoAlvo", Convert.ToDecimal(alvo1.DistanciaDoCanhaoAlvo));
            //Referencia o atributo da SP de "DistanciaDoCanhao" como o atributo de "DistanciaDoCanhao" do objeto Alvo

            comandoInseridosTabelaAlvo.Parameters.AddWithValue("@VelocidadeInicialAlvo", Convert.ToDecimal(alvo1.VelocidadeAlvo));
            //Referencia o atributo da SP de "VelocidadeInicialAlvo" como o atributo de "VelocidadeAlvo" do objeto Alvo

            comandoInseridosTabelaAlvo.Parameters.AddWithValue("@AnguloMaximoAcertoAlvo", Convert.ToDecimal(alvo1.AnguloMaximo));
            //Referencia o atributo da SP de "AnguloMaximoAcertoAlvo" como o atributo de "AnguloMaximo" do objeto Alvo

```

```

comandoInseredadosTabelaAlvo.Parameters.AddWithValue("@TempoateAtingirSoloAlvo", convert.ToDecimal(alvo1.TempoateAtingirSoloAlvo));
//Referencia o atributo da SP "Tempoateatingirsoloalvo" como o atributo de "Tempoateatingirsoloalvo" do objeto Alvo

//Realiza a execução do comando como uma execução de uma "Não Query", tratando-se de uma SP
comandoInseredadosTabelaAlvo.ExecuteNonQuery();

}

}

//Criando método que cria a SP que insere os dados na tabela de simulação
1 referência
internal void criaSPsimulacao()
{
    //Realiza o reacionamento e referencia da conexão com o banco de dados com o valor contido na variavel 'endereçoBancoDeDados' no programa de execução principal do software,
    //evocado anteriormente em 'referenciaClassePrograma'
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.endereçoBancoDeDados))
    {
        //Abrindo conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Criando SP
        string criandoSPInclusaoDadosSimulacao = @"
--Declarando nome e estado de criação ou alteração da SP
CREATE OR ALTER PROCEDURE InserirSimulacao

--Declarando parametros da SP e suas tipagem de dados
@AnguloTrajetoriaProjtil DECIMAL(10,2),
@VelocidadeInicial1Projtil DECIMAL(10,2),
@VelocidadeInicial2Projtil DECIMAL(10,2),
@VelocidadeusadaProjtil DECIMAL(10,2),
@TempovoProjtil DECIMAL(10,2),
@TipoMovimentoProjtil VARCHAR(30),
@ExistenciaResolucaoRealAnguloFornecido VARCHAR(3),
@AlturaInicialAlvo DECIMAL(10,2),
@DistanciaDoCanhaoAlvo DECIMAL(10,2),
@VelocidadeInicialAlvo DECIMAL(10,2),
@AnguloMaximoAcertoAlvo DECIMAL(10,2),
@Tempoateatingirsoloalvo DECIMAL(10,2)

AS
1
BEGIN

```

```

--Declarando variaveis e suas tipagem de dados
DECLARE @ProximoID INT;
DECLARE @IDProjtil INT;
DECLARE @IDAlvo INT;

--Declarando o valor que será recebido pela variavel 'ProximoID'. Essa variavel seleciona o próximo ID de simulação, selecionando o número máximo disponível na tabela e adiciona 1 (um)
SELECT @ProximoID = ISNULL(MAX(id_simulacao), 0) + 1 FROM simulacoes;

--Declarando o valor que será recebido pela variavel 'IDProjtil'. Essa variavel seleciona o ID máximo do projtil
SELECT @IDProjtil = MAX(id_projtil) FROM projtil;

--Declarando o valor que será recebido pela variavel 'IDAlvo'. Essa variavel seleciona o ID máximo do alvo
SELECT @IDAlvo = MAX(id_alvo) FROM alvo;

--Define os campos da tabela que irão receber os valores e quais parametros da SP irão ser incluídos nos campos referenciados
INSERT INTO simulacoes (id_simulacao,
id_projtil,
id_alvo,
angulo_trajetoria_projtil,
velocidade_inicial1_projtil,
velocidade_inicial2_projtil,
velocidade_usada_projtil,
tempo_de_voo_projtil,
tipo_movimento_projtil,
existencia_resolucao_real_angulo_fornecido,
altura_inicial_alvo,
distancia_canhao_alvo,
velocidade_inicial_alvo,
angulo_maximo_acerto_alvo,
tempo_ate_attingir_solo_alvo)

VALUES (@ProximoID,
@IDProjtil,
@IDAlvo,
@AnguloTrajetoriaProjtil,
@VelocidadeInicial1Projtil,
@VelocidadeInicial2Projtil,
@VelocidadeusadaProjtil,
@TempovoProjtil,
@TipoMovimentoProjtil,
@ExistenciaResolucaoRealAnguloFornecido,
@AlturaInicialAlvo,
@DistanciaDoCanhaoAlvo,
@VelocidadeInicialAlvo,
@AnguloMaximoAcertoAlvo,
@Tempoateatingirsoloalvo);

END";

//Cria instancia Sql que realiza a execução do comando de criação ou alteração da SP
using (SqlCommand comandoQuerySPInseredadosSimulacao = new SqlCommand(criandoSPInclusaoDadosSimulacao, conexaoBancoDeDados))
{
    comandoQuerySPInseredadosSimulacao.ExecuteNonQuery();
}

```

```

//Cria método que cadastra uma nova simulação
1 referência
internal void cadastraSimulacao()
{
    //Abre conexão com o banco de dados com o endereçamento referenciado no momento de instancia do programa
    using (SqlConnection conexaoBancoDeDados = new SqlConnection(referenciaClassePrograma.enderecamentoBancoDeDados))
    {
        //Abre conexão com o banco de dados
        conexaoBancoDeDados.Open();

        //Inicia as variáveis de comando com o Stored Procedure de "InserirAlvo"
        using (SqlCommand comandoInserirDadosTabelaSimulacao = new SqlCommand("InserirSimulacao", conexaoBancoDeDados))
        {
            //Referencia o comando de inserir dados na tabela de alvo como uma SP
            comandoInserirDadosTabelaSimulacao.CommandType = CommandType.StoredProcedure;

            //Atribui os valores de referencia na SP de "Inserir Alvo"
            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@AnguloTrajetoriaProjtil", Convert.ToDecimal(projtil1.AnguloLancamentoProjtil));
            //Converte o valor de angulo do lançamento do projtil para decimal e atribui ele ao parametro correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@VelocidadeInicial1Projtil", Convert.ToDecimal(projtil1.VelocidadeInicial1Projtil));
            //Converte o valor de velocidade inicial 1 do projtil para decimal e atribui ele ao parametro correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@VelocidadeInicial2Projtil", Convert.ToDecimal(projtil1.VelocidadeInicial2Projtil));
            //Converte o valor de velocidade inicial 2 do projtil para decimal e atribui ele ao parametro correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@VelocidadeUsadaProjtil", Convert.ToDecimal(projtil1.VelocidadeUsada));
            //Converte o valor de velocidade usada do projtil para decimal e atribui ele ao parametro correspondente a SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@TempoVooProjtil", Convert.ToDecimal(projtil1.TempoVoo));
            //Converte o valor de tempo de voo do projtil para decimal e atribui ele ao parametro correspondente a SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@TipoMovimentoProjtil", projtil1.TipoMovimento);
            //Atribui o valor do tipo de movimento do projtil para o parametro correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@ExistenciaResolucaoRealAnguloFornecido", projtil1.ResolucaoRealParaAngulo);
            //Atribui o valor do conceito de resolução real para o ângulo, para o parametro correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@AlturaInicialAlvo", Convert.ToDecimal(alvo1.AlturaInicialAlvo));
            //Converte o valor da altura inicial do alvo para decimal e atribui ele para o valor correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@DistanciaDoCanhaoAlvo", Convert.ToDecimal(alvo1.DistanciaDoCanoaAlvo));
            //Converte o valor a distancia do alvo até o canhão para decimal e atribui ele para o valor correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@VelocidadeInicialAlvo", Convert.ToDecimal(alvo1.VelocidadeAlvo));
            //Converte o valor da velocidade do alvo para decimal e atribui ele ao parametro correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@AnguloMaximoAcertoAlvo", Convert.ToDecimal(alvo1.AnguloMaximo));
            //Converte o valor do ângulo máximo do alvo para decimal e atribui ele ao parametro correspondente na SP

            comandoInserirDadosTabelaSimulacao.Parameters.AddWithValue("@TempoAtéatingirSoloAlvo", Convert.ToDecimal(alvo1.TempoAtéatingirSoloAlvo));
            //Converte o valor do tempo que decorre até que o alvo atinga o solo para decimal e atribui ele ao parametro correspondente na SP
        }

        //Realiza a execução do comando como uma execução de uma "Não Query", tratando-se de uma SP
        comandoInserirDadosTabelaSimulacao.ExecuteNonQuery();
    }
}

//Cria método que escreve o arquivo CSV que contém os dados do gráfico
1 referência
private void EscreverArquivoCSV(string nome_arquivo) //Define como parametro receptor do método o caminho do arquivo que será criado
{
    //Define o nome da pasta que será criada
    string nomeSubpasta = "PBLGeracaoGrafico";

    //Captura o caminho da pasta padrão de 'documentos' na qual o software está sendo executado
    string pastaDocumentos = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

    //Combina os caminhos declarados anteriormente e determina o caminho final de uma pasta de destino
    string pastaDestino = Path.Combine(pastaDocumentos, nomeSubpasta);

    //Julga que caso o diretório a ser construído não exista, ele então é criado do zero
    if (!Directory.Exists(pastaDestino + "\\\" + nome_arquivo))
    {
        Directory.CreateDirectory(pastaDestino);
    }

    //Cria um caminho de salvamento para o arquivo, onde será combinado o caminho da pasta destino e o nome do arquivo
    string caminhoArquivo = Path.Combine(pastaDestino, nome_arquivo);

    //Escreve um arquivo vazio com o nome declarado
    File.WriteAllText(caminhoArquivo, string.Empty);

    //Caso a pasta de arquivo e o arquivo não exista, retorna uma caixa de mensagem informando que um erro ocorreu
    if (!File.Exists(pastaDestino + "\\\" + nome_arquivo))
    {
        MessageBox.Show("Erro ao localizar o arquivo de input");
    }

    //Calcula parametros com definições de cinco vezes maiores que os obtidos
    //Calcula o tempo máximo de voo
    double maximo_tempo_voo = projtil1.TempoVoo * 5;

    //Calcula a posição máxima em y do projtil
    double maximo_posicao_em_y_projtil = projtil1.VelocidadeInicialEmVProjtil * maximo_tempo_voo - (4.9 * Math.Pow(maximo_tempo_voo, 2));
    maximo_posicao_em_y_projtil = Math.Round(maximo_posicao_em_y_projtil, 2); //Arredonda o valor em duas casas decimais
}

```

```

//Calcula a posição máxima em y do alvo
double maximo_posicao_em_y_alvo = alvoi.AlturainicialAlvo - alvoi.VelocidadeAlvo * maximo_tempo_voo;

//Define o nome das colunas que serão encontradas no arquivo
string adicionarConteudoColunas = "TempoDeVooProjtil;MaximoTempoDeVooProjtil;PosicaoEmProjtil;AlturainicialAlvo;MaximoPosicaoEmProjtil;PosicaoEmAlvo;MaximoPosicaoEmAlvo";

//Define o conteúdo das colunas
string adicionarConteudoDados = $"{Math.Abs(projetili.TempoVoo)};{Math.Abs(maximo_tempo_voo)};{Math.Abs(projetili.PosicaoEmProjtil)};{Math.Abs(alvo.AlturainicialAlvo)};" +
    $"{Math.Abs(maximo_posicao_em_y_projtil)};{Math.Abs(alvo.PosicaoEmAlvo)};{Math.Abs(maximo_posicao_em_y_alvo)}";

//Une os conteúdos das colunas em um esquema de gravação no arquivo
File.AppendAllText(pastaDestino + "\\\" + nome_arquivo, adicionarConteudoColunas + Environment.NewLine);

//Une os conteúdos digitados e referenciados em um esquema de gravação de arquivo
File.AppendAllText(pastaDestino + "\\\" + nome_arquivo, adicionarConteudoDados + Environment.NewLine);
}

//Criando método que realiza a execução da abertura do código em python que gera o arquivo da simulação
1 referencia
private void AbreScriptPython()
{
    //O CAMINHO DO SCRIPT EM PYTHON DEVE SER ALTERADO CONFORME A MÁQUINA NA QUAL O SOFTWARE ESTÁ SENDO EXECUTADA
    Process.Start("python", @"D:\Vitor\FESA\PL\VP_L\PythonScripts\GeracaoGrafico.py");
}

//Cria método que realiza o cálculo da trajetória
1 referencia
private void RealizaCalculo()
{
    //Verifica o tipo de dado no qual os inputs fornecidos nas textBox's existentes no visual do WindowsForms. Todos devem receber dados do tipo double
    if (double.TryParse(textBoxDadosAlvoAlturaInicial.Text, out double valorTextBoxDadosAlvoAlturaInicial) &&
        double.TryParse(textBoxDadosAlvoDistanciaDoCanhao.Text, out double valorTextBoxDadosAlvoDistanciaDoCanhao) &&
        double.TryParse(textBoxDadosVelocidadeInicialAlvo.Text, out double valorTextBoxDadosAlvoVelocidadeInicial) &&
        double.TryParse(textBoxDadosProjtilAnguloTrajetoria.Text, out double valorTextBoxDadosProjtilAnguloTrajetoria))
    {
        //Atribuindo valores iniciais
        projetili.AnguloLancamentoProjtil = valorTextBoxDadosProjtilAnguloTrajetoria;
        alvoi.AlturainicialAlvo = valorTextBoxDadosAlvoAlturaInicial;
        alvoi.DistanciaDoCanhaoAlvo = valorTextBoxDadosAlvoDistanciaDoCanhao;
        alvoi.VelocidadeAlvo = valorTextBoxDadosAlvoVelocidadeInicial;

        //Realizando o cálculo do tempo que decorre até que o alvo atinja o solo
        alvoi.CalcularTempoAttingirSoloAlvo();

        //Realizando o cálculo do ângulo máximo
        alvoi.CalcularAnguloMaximo();

        //Realizando condição para visualizar atendimento do ângulo digitado pelo usuário, compreenda as variações do ângulo máximo
        projetili.AnguloLancamentoProjtil = (projetili.AnguloLancamentoProjtil) * Math.PI / 180.0;
    }
}

```

```

//Verifica as condições onde o ângulo do lançamento do projetil fornecido pelo usuário e convertido em radianos seja menor ou igual ao ângulo máximo e seja maior que '0'
if ((projetili.AnguloLancamentoProjtil <= alvoi.AnguloMaximo) &&
    (projetili.AnguloLancamentoProjtil > 0.00))
{
    //Resolvendo equação do lançamento
    valoresCalculoi.VisaoA = alvoi.DistanciaDoCanhaoAlvo * Math.Tan(projetili.AnguloLancamentoProjtil) - alvoi.AlturainicialAlvo; //Resolvendo valor em A
    valoresCalculoi.VisaoB = alvoi.VelocidadeAlvo * alvoi.DistanciaDoCanhaoAlvo / (CodigoMain.MetodoCosseno.CalcularCosseno(projetili.AnguloLancamentoProjtil)); //Resolvendo valor em B
    valoresCalculoi.VisaoC = -4.9 * (Math.Pow(alvoi.DistanciaDoCanhaoAlvo, 2)) / Math.Pow((CodigoMain.MetodoCosseno.CalcularCosseno(projetili.AnguloLancamentoProjtil)), 2); //Resolvendo valor em C

    //Resolvendo delta da equação
    valoresCalculoi.CalcularDiscriminanteDelta();

    // Realizando condicionamento para o valor de delta encontrado
    //Caso o valor de delta seja menor ou igual a zero, retorna uma message Box informando ao usuário que não existe uma resolução real para o ângulo digitado
    if (valoresCalculoi.DiscriminanteDelta <= 0)
    {
        //Exibe uma caixa de mensagem para o usuário informando valores sobre o seu ângulo digitado
        MessageBox.Show("Infelizmente, não há uma resolução real para o ângulo fornecido \n" + //Informa ao usuário que o ângulo digitado não possui resolução real
            "Valor em A: " + Math.Round(valoresCalculoi.VisaoA, 2) + "\n" + //Informa o valor do coeficiente em A
            "Valor em B: " + Math.Round(valoresCalculoi.VisaoB, 2) + "\n" + //Informa o valor do coeficiente em B
            "Valor em C: " + Math.Round(valoresCalculoi.VisaoC, 2) + "\n" + //Informa o valor do coeficiente em C
            "Valor de Delta: " + Math.Round(valoresCalculoi.DiscriminanteDelta, 2)); //Informa o valor de Delta

        //Limpa os valores de todos os TextBoxes presentes na tela
        textBoxDadosAlvoAlturaInicial.Text = "";
        textBoxDadosAlvoDistanciaDoCanhao.Text = "";
        textBoxDadosProjtilAnguloTrajetoria.Text = "";
        textBoxDadosVelocidadeInicialAlvo.Text = "";

        //Realiza o cadastro do projetil simulado com os dados definidos
        projetili.VelocidadeInicialProjtil = 0.00;
        projetili.VelocidadeInicial2Projtil = 0.00;
        projetili.TempoVoo = 0.00;
        projetili.TipoMovimento = "N/A";
        projetili.ResolucaoRealParaAngulo = "N";
        cadastraProjtil();

        //Realiza o cadastro do alvo com os dados definidos
        cadastraAlvo();

        //Realiza o cadastro da simulação completa
        cadastraSimulacao();
    }
}

```

```

//Caso exista uma resolução real para o ângulo, o cálculo é iniciado
else
{
    //Define um valor para o atributo de que se existe resolução real no ângulo
    projetili.ResolucaoRealParaAngulo = "S";

    //Calcula o valor de velocidade inicial 1
    projetili.VelocidadeInicialProjtil = (-valoresCalculoi.VisaoB + Math.Sqrt(valoresCalculoi.DiscriminanteDelta)) / (2 * valoresCalculoi.VisaoA);

    //Calcula o valor de velocidade inicial 2
    projetili.VelocidadeInicial2Projtil = (-valoresCalculoi.VisaoB - Math.Sqrt(valoresCalculoi.DiscriminanteDelta)) / (2 * valoresCalculoi.VisaoA);

    //Decide qual velocidade deve ser usada
    projetili.VelocidadeUsada = projetili.VelocidadeInicialProjtil > 0 ? projetili.VelocidadeInicial1Projtil : projetili.VelocidadeInicial2Projtil;

    //Analisar condição para determinar o tipo de movimento do projetil
    projetili.TipoMovimento = alvoi.VelocidadeAlvo > (projetili.VelocidadeUsada * CodigoMain.MetodoSeno.CalcularSeno(projetili.AnguloLancamentoProjtil) -
        4.9 * Math.Pow(projetili.TempoVoo, 2)) ? "Ascendente" : "Descendente";

    //Calcula a velocidade do projetil em X
    projetili.VelocidadeInicialXProjtil = projetili.VelocidadeUsada * CodigoMain.MetodoCosseno.CalcularCosseno(projetili.AnguloLancamentoProjtil);

    //Calcula a velocidade do projetil em Y
    projetili.VelocidadeInicialYProjtil = projetili.VelocidadeUsada * CodigoMain.MetodoSeno.CalcularSeno(projetili.AnguloLancamentoProjtil);

    //Define uma posição inicial para o projetil
    projetili.PosicaoEmProjtil = 0.00;

    //Criando modelo de equação quadrática para encontrar o tempo de encontro entre projetil e alvo
    double equacaoEncontroA = 4.9; //Resolvendo valor de A
    double equacaoEncontroB = -(projetili.VelocidadeInicialYProjtil + alvoi.VelocidadeAlvo); //Resolvendo valor em B
    double equacaoEncontroC = alvoi.AlturainicialAlvo; //Resolvendo valor em C

    //Calcula o valor de Delta
    double discriminanteDelta = Math.Pow(equacaoEncontroB, 2) - 4 * equacaoEncontroA * equacaoEncontroC;
}

```

```

//Verifica se o valor de delta é maior que zero
if (discriminanteEmDelta > 0)
{
    //Obtem os valores de tempo 1 e tempo 2, usando calculos padrões de formula de Baskhara
    double tempo1 = (-equacaoEncontroB + Math.Sqrt(discriminanteEmDelta)) / (2 * equacaoEncontroA);
    double tempo2 = (-equacaoEncontroB - Math.Sqrt(discriminanteEmDelta)) / (2 * equacaoEncontroA);

    //Atribui um valor para o atributo de 'Tempovoo' do projetil, selecionando a menor variação entre esses.
    projetil1.Tempovoo = Math.Min(tempo1, tempo2);
    projetil1.Tempovoo = Math.Round(projetil1.Tempovoo, 2); //Arredonda o valor em até duas casas decimais após a vírgula

    //Calcula a posição em Y para o projetil
    projetil1.PosicaoEmProjetil = projetil1.VelocidadeInicialEmProjetil * projetil1.Tempovoo - (4.9 * Math.Pow(projetil1.Tempovoo, 2));
    projetil1.PosicaoEmProjetil = Math.Round(projetil1.PosicaoEmProjetil, 2); //Arredonda o valor em até duas casas decimais após a vírgula

    //Calcula a posição em Y para o alvo
    alvo1.PosicaoEmAlvo = alvo1.AlturaInicialAlvo - alvo1.VelocidadeAlvo * projetil1.Tempovoo;
    alvo1.PosicaoEmAlvo = Math.Round(alvo1.PosicaoEmAlvo, 2); //Arredonda o valor em até duas casas decimais após a vírgula
}

//Caso a verificação falhe, define 0 para o atributo de "Tempovoo" do projetil
else
{
    projetil1.Tempovoo = 0.00;
}

//Exibe os resultados finais em uma caixa de mensagem
MessageBox.Show("confira os resultados abaixo \n" +
    "Valor em A: " + Math.Round(valoresCalculo1.VisaoA, 2) + "\n" +
    //Informa o valor em A

    "Valor em B: " + Math.Round(valoresCalculo1.VisaoB, 2) + "\n" +
    //Informa o valor em B

    "Valor em C: " + Math.Round(valoresCalculo1.VisaoC, 2) + "\n" +
    //Informa o valor em C

    "Valor de Delta: " + Math.Round(valoresCalculo1.DiscriminanteEmDelta, 2) + "\n\n" +
    //Informa o valor de Delta

    "Velocidade inicial do projetil sugerida: " + Math.Round(projetil1.VelocidadeUsada, 2) + " m/s \n" +
    //Informa o valor da velocidade inicial sugerida para o projetil

    "Intervalo do tempo de voo: " + Math.Round(projetil1.Tempovoo, 2) + " segundos \n" +
    //Informa o valor em que o projetil atingiria o alvo

    "Tipo de movimento do acerto: " + projetil1.TipoMovimento);
    //Informa o tipo de acerto

CadastraProjetil(); //Executa o método de cadastrar projetil

```

```

CadastraAlvo(); //Executa método de cadastrar alvo
CadastraSimulacao(); //Executa método de cadastrar simulacao
EscreverArquivoCSV("dados_simulacao.txt"); //Executa método que grava arquivo dos dados obtidos
AbreScriptPython(); //Executa método que executa script em python que gera o gráfico de simulacao
}

//Caso a verificação falhe, exibe uma caixa de mensagem e apaga todo conteúdo da textBox de angulo referenciado
else
{
    MessageBox.Show("Não existe possibilidades para o ângulo digitado. Por favor, digite outro angulo");
    textBoxDadosProjetilAnguloTrajetoria.Text = "";
}
}

//Define ações que serão executadas após o clique do botão 'buttonSimularTelaSimulacao'
1 referência
private void buttonSimularTelaSimulacao_Click(object sender, EventArgs e)
{
    CriaSPCadastraProjetil(); //Executa método que cria ou altera SP de cadastro de projetil
    CriaSPCadastraAlvo(); //Executa método que cria ou altera SP de cadastro de alvo
    CriaSPSimulacao(); //Executa método que cria ou altera SP de cadastro de simulacao

    //Executa método que realiza o calculo
    RealizaCalculo();
}

//Define ações que serão executadas após o clique do botão 'buttonObterAnguloIdeal'
1 referência
private void buttonObterAnguloIdeal_Click(object sender, EventArgs e)
{
    //Realiza checagem do tipo de dado digitado nas textBox's
    if (double.TryParse(textBoxDadosAlvoAlturaInicial.Text, out double valorTextBoxDadosAlvoAlturaInicial) &&
        double.TryParse(textBoxDadosAlvoDistanciaDoCanhao.Text, out double valorTextBoxDadosAlvoDistanciaDoCanhao) &&
        double.TryParse(textBoxDadosVelocidadeInicialAlvo.Text, out double valorTextBoxDadosAlvoVelocidadeInicial))
    {
        //Atribui valores iniciais para os valores digitados
        alvo1.AlturaInicialAlvo = valorTextBoxDadosAlvoAlturaInicial;
        alvo1.DistanciaDoCanhaoAlvo = valorTextBoxDadosAlvoDistanciaDoCanhao;
        alvo1.VelocidadeAlvo = valorTextBoxDadosAlvoVelocidadeInicial;
    }
}

```

```

        //Executa método de angulo máximo
        alvo.CalculaAnguloMaximo();

        //Exibe caixa de mensagem que mostra para o usuário, a partir de qual ângulo o mesmo deve digitar
        MessageBox.Show("Para os valores de alvo digitados, o angulo de lançamento mínimo do projétil deve ser a partir de " + alvo.AnguloMaximo);

    }

    //Caso a verificação retorne status falso (False), exibe uma caixa de mensagem que solicita ao usuário que digite valores corretos
    else
    {
        MessageBox.Show("Valores digitados incorretamente, por favor, forneça valores corretos");
    }
}

1 referencia
private void TelaSimulacao_Load(object sender, EventArgs e)
{
}

1 referencia
private void label7_Click(object sender, EventArgs e)
{
}

//Define ações que serão realizadas após o clique do botão 'buttonSairTelaSimulacao'
1 referencia
private void buttonSairTelaSimulacao_Click(object sender, EventArgs e)
{
    //cria novo objeto que referencia windowsforms atual
    Form telaAtual = Form.ActiveForm;

    //Cria novo objeto que referencia a 'TelaMenu', atribuindo para uma variavel de nome de 'telaMenu'
    TelaMenu telaMenu = new TelaMenu();

    //Exibe uma nova 'TelaMenu'
    telaMenu.Show();

    //Fecha tela atual
    telaAtual.Close();
}
}

```

- TELA SIMULAÇÃO (DESING)

Realizar simulacao

● Dados para o alvo ●

Altura inicial metros(m)

Distância do canhão metros(m)

Velocidade Inicial metros por segundo (m/s)

Obter o ângulo ideal

Dados para o projétil

Angulo da Trajetoria ° graus

Para valores decimais, utilize virgulas | Exemplo: 75.01

Simular!

Sair

```

namespace PPL_Main.CodigoMain
{
    2 references
    public class ValoresCalculoQuacaoQuadratica
    {
        private double visaoA;
        private double visaoB;
        private double visaoC;
        private double discriminanteDelta;

        6 references
        public double VisaoA
        {
            get { return visaoA; }
            set { visaoA = value; }
        }

        6 references
        public double VisaoB
        {
            get { return visaoB; }
            set { visaoB = value; }
        }

        4 references
        public double VisaoC
        {
            get { return visaoC; }
            set { visaoC = value; }
        }

        6 references
        public double DiscriminanteDelta
        {
            get { return discriminanteDelta; }
            set { discriminanteDelta = value; }
        }

        1 reference
        public void CalculaDiscriminanteDelta()
        {
            DiscriminanteDelta = Math.Pow(visaoB, 2) - 4 * visaoA * VisaoC;
        }
    }
}

```

```

namespace PPL_Main.Telas
{
    4 references
    partial class TelaSimulacao
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        6 references
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows Form Designer generated code
        private System.Windows.Forms.Label labelDadosParaAlvo;
        private System.Windows.Forms.Label labelDadosAlvoAlturaInicial;
        private System.Windows.Forms.TextBox textBoxDadosAlvoAlturaInicial;
        private System.Windows.Forms.Label labelDadosAlvoVelocidadeInicial;
        private System.Windows.Forms.Label labelDadosAlvoDistanciaCanhao;
        private System.Windows.Forms.Label labelDadosAlvoVelocidadeCanhao;
        private System.Windows.Forms.TextBox textBoxDadosAlvoDistanciaCanhao;
        private System.Windows.Forms.Label labelDadosAlvoVelocidadeCanhao;
        private System.Windows.Forms.Label labelDadosProjetoGaussAnguloTrajetoria;
        private System.Windows.Forms.TextBox textBoxDadosProjetoGaussAnguloTrajetoria;
        private System.Windows.Forms.Label labelDadosProjetoGaussAnguloTrajetoria;
        private System.Windows.Forms.Button buttonSimularTelaSimulacao;
        private System.Windows.Forms.BindingSource pBLDataSetBindingSource;
        private System.Windows.Forms.BindingSource pBLDataSetBindingSource1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox textBoxDadosVelocidadeInicialAlvo;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Button buttonObterAnguloIdeal;
        private System.Windows.Forms.BindingSource pBLDataSetBindingSource2;
        private System.Windows.Forms.BindingSource pBLDataSetBindingSource3;
        private System.Windows.Forms.BindingSource pBLDataSetBindingSource4;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Button buttonSairTelaSimulacao;
    }
}

```

- APLICAÇÃO DO SOFTWARE PARA RESOLVER AS EQUAÇÕES OBTIDAS E VALIDAÇÃO DO SISTEMA

Para um melhor atendimento da aplicação do software desenvolvido, validação do sistema e resolução da situação problema proposta, favor assistir o vídeo que foi disponibilizado junto a entrega desse relatório ou [acessar o link no qual o vídeo encontra-se publicado](#).

Caso seja do interesse realizar a execução do software em uma máquina local, favor, assistir também o vídeo de tutorial de alterações de parâmetros do código que foi disponibilizado junto a entrega desse relatório, ou [conferir através do link](#) no qual ele está

publicado. Sugerimos também conferir o documento de tutorial de instruções de instalações de softwares e módulos auxiliares para funcionamento correto e completo do software que foi disponibilizado junto a entrega desse relatório, ou [conferir o mesmo através do link](#).