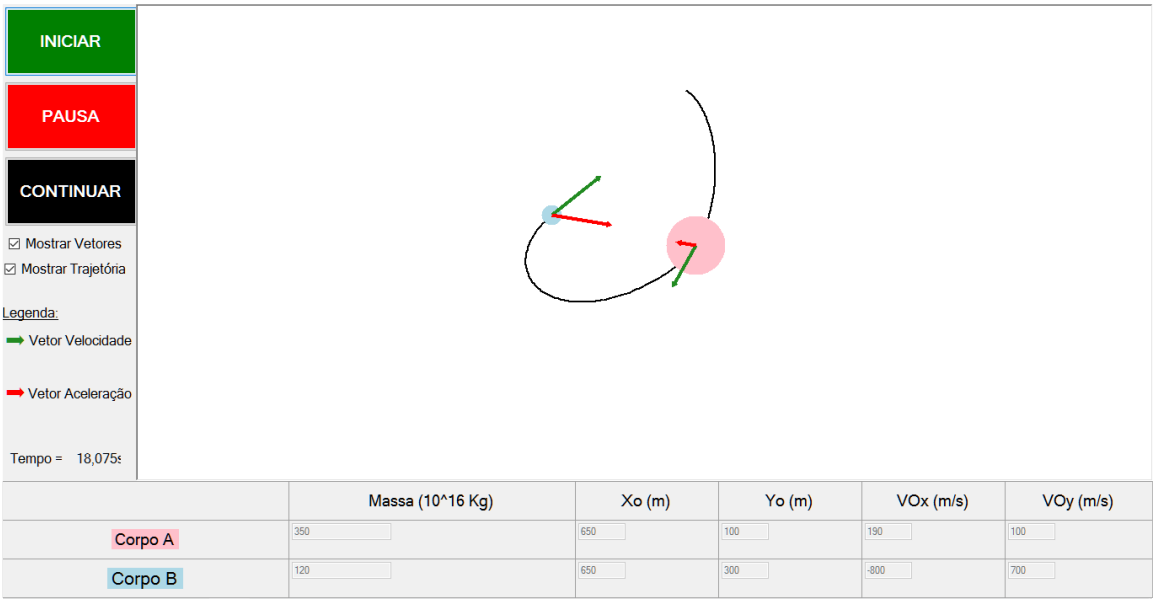
ESCOLA SECUNDÁRIA D. INÊS DE CASTRO

CURSO CIENTÍFICO-HUMANÍSTICO DE

CIÊNCIAS E TECNOLOGIAS

PROJETO DE APLICAÇÕES INFORMÁTICAS B

Simulador Físico: Força Gravítica



DIANA RIBEIRO **Nº:**10

FRANCISCO SERRALHEIRO **Nº:**14

ANO: 12º **TURMA:** CT-A

PROFESSOR: Nuno José da Silva Trindade Duarte

12-12-2016

Índice

1. Resumo

Este relatório descreve o processo e o trabalho que esteve subjacente à criação de uma aplicação. Essa aplicação tem como principal objetivo simular o movimento de dois corpos que estão em interação gravítica entre si, apenas com as equações clássicas (horárias) da cinemática.

Começámos por tratar dos problemas gráficos e fomos evoluindo para os problemas físicos da simulação, incluindo, em ambos, as "traduções" e resoluções programáticas. Optámos por resolver problema a problema, concentrando-nos num de cada vez. No que toca aos problemas gráficos, aprofundámos o problema da interface do Simulador, o problema do desenho dos corpos, o problema do desenho dos vetores físicos e o problema do desenho da trajetória. Relativamente aos problemas físicos, aprofundámos mais o problema do movimento dos corpos e da colisão entre estes.

Terminámos este período com uma aplicação que retrata, razoavelmente, o movimento orbital, sendo digna de uso em qualquer sala de aula de Física ou para estudo pessoal.

2. Introdução

O presente relatório é elaborado no âmbito da disciplina de Aplicações Informáticas B, com vista à conclusão do projeto proposto na unidade 1-Introdução à programação.

O projeto desenvolveu-se na Escola Secundária D. Inês de Castro, nas aulas da disciplina, durante todo o período, em linguagem de programação C#, com o apoio do programa Visual Studio.

O relatório destina-se, não só a descrever os problemas surgidos ao longo do trabalho, como também a apresentar um enquadramento do projeto realizado com todo o conhecimento adquirido durante o mesmo.

Este relatório está dividido em três partes. Na primeira é feita uma breve descrição do problema, bem como uma apresentação dos recursos utilizados. A segunda parte tem como finalidade expor o desenvolvimento do projeto, isto é, explicar todos os problemas gráficos, físicos e programáticos com que nos deparámos ao longo do período. A terceira parte contém a conclusão, a reflexão final, as referências e os anexos.

Ambos temos como disciplina de opção Física e ambos gostamos muito da área, pelo que o tema deste relatório é “Simulador Físico: Força Gravítica”, que consiste num simulador do movimento entre dois corpos que estão sob o efeito da Força Gravítica, como, por exemplo, acontece no vácuo. Este projeto teve como grande vantagem a oportunidade de entrar em contacto com o mundo da programação e da física, simultaneamente, de forma a complementar e aperfeiçoar as competências tão procuradas nos dias de hoje, onde a tecnologia prevalece. Além disso, podemos utilizar esta aplicação quer para estudo pessoal quer para, se possível, dar suporte aos professores nas suas aulas de Física.

3. Descrição do problema

O tema inicial do projeto que se pretendeu resolver, ao longo do período, é um Simulador Físico, que simulava o movimento orbital entre dois corpos celestes. O problema inicial a que nos propusemos a resolver teve como base a questão: "Será que é possível desenvolver uma aplicação que simule o movimento causado pela força gravítica entre dois corpos celestes, apenas com equações (clássicas) do movimento?".

Assim, pretendemos atingir uma aplicação sem erros científicos, com uma animação suave, e apta de assistir qualquer professor ou aluno de Física.

4. Recursos utilizados

Neste projeto utilizámos variados recursos/ferramentas para o suporte e o desenvolvimento da aplicação. Primeiramente, optámos pelo Visual Studio Community 2015, na versão 14.0, como ambiente de programação. Para manter o trabalho atualizado entre os diversos aparelhos usámos pens USB e o Google Drive. O grupo de Facebook de Aplicações Informáticas B serviu como meio de comunicação entre os colegas e o professor orientador. Tivemos ainda o contributo de entidades externas, como o professor da disciplina, e a Escola Secundária D. Inês de Castro, que nos forneceu o material *hardware* para a realização do trabalho.

5. Desenvolvimento do projeto

No decorrer do projeto fomo-nos apercebendo de algumas dificuldades, muitas delas corrigidas, outras contornadas, mas foi sobretudo um desenvolvimento suave e sem grandes atribulações.

Desde já, optámos por criar uma classe que representava os corpos celestes à qual chamámos "Corpo" e outra classe que representava qualquer vetor na aplicação (aceleração ou velocidade) à qual denominámos "Vector2D". Decidimos, portanto, trabalhar com programação orientada a objetos, porque achámos que, para o trabalho em vista, seria a melhor opção.

5.1. Problemas Gráficos

Inicialmente havia no ar diversos problemas de programação, maioritariamente relacionados com as equações físicas a aplicar para a construção do movimento orbital. Devido a isso, começámos por nos concentrar nos problemas gráficos:

- Problema da interface do Simulador: Para resolver este problema tomámos como inspiração a interface do simulador de órbitas e gravidade da universidade do Colorado. Num painel gráfico que cobre, cerca de 75% do *Form* é onde se vai desenhar a simulação. Nas suas extremidades estão os botões "INICAR", "PAUSAR" e "CONTINUAR" para o utilizador controlar o movimento à sua preferência. Tem também duas *checkboxes* para monitorizar a visualização dos vetores e/ou da trajetória. E finalmente, na extremidade inferior encontra-se uma tabela com as grandezas físicas (Massas, Posições e Velocidades) para o utilizador regular o movimento que vai simular.
- Problema do desenho dos corpos: Na classe "Corpo", designámos um método que desenhava os corpos (sem vetores aceleração e velocidade). Primeiramente, pensámos em desenhar elipses, com *height* e *width* de igual valor (para formar um círculo), em que estes valores seriam diretamente iguais aos valores da massa atribuídos pelo utilizador. No entanto, reparámos que esta solução não era a mais adequada pois um valor muito grande preencheria todo o painel de desenho. Portanto, no que toca ao desenho do corpo, declarámos uma variável na classe do Corpo, à qual chamámos "*DrawingRadius*". Era esta variável que iria determinar o raio do desenho do círculo (representando o corpo celeste), cujo valor seria 10% do valor da massa inserido pelo utilizador.
- Problema do desenho dos vetores aceleração e velocidade: Para resolver este problema definimos um método na classe "Corpo" que desenhava os corpos com os vetores. O ponto de aplicação dos vetores teria de ser desenhado nos centros de massa dos corpos. Para isso, tivemos de arranjar maneira de igualar o canto superior esquerdo do retângulo da elipse ao centro de massa, cuja solução foi encontrada subtraindo à posição do corpo desenhado o valor do *DrawingRadius*. O ponto final dos vetores velocidade e aceleração seria a soma entre

as componentes X e Y do centro de massa e as componentes X e Y da velocidade e da aceleração, cujos valores vão sendo alterados ao longo do movimento. Posto isto, deparámo-nos com o problema do tamanho dos vetores, pois quando iniciávamos a aplicação reparávamos que o comprimento destes era enorme, o que faz sentido, porque se o utilizador colocasse, por exemplo, 400 m/s na velocidade inicial, independentemente da componente (X ou Y), a sua dimensão não era apropriada para a proporção do painel gráfico. Deste modo, vimo-nos forçados a dividir os comprimentos gráficos. No vetor velocidade por 10 e no vetor aceleração por 100, não havendo qualquer relação entre estes.

- Problema do desenho da trajetória: Uma vez que uma das opções fornecidas ao utilizador é a de escolher se quer ou não visualizar a trajetória e/ou os vetores, em primeiro lugar, tivemos que conceber uma série de condições para colocar as *checkboxes* a funcionar corretamente. Tivemos várias ideias para desenhar a trajetória. A primeira foi a de desenhar um círculo, com raio minúsculo, e centro igual à posição do corpo. Não havendo método *Clear*() para apagar o círculo, o seu desenho arrastar-se-ia, resultando numa trajetória. Outra ideia seria desenhar uma linha entre a posição atual e a posição anterior dos corpos. No entanto, ambas eram impossíveis de executar, uma vez que o método *Clear*() utilizado para apagar os corpos era necessário a cada *tick* do *timer* da animação, e esse mesmo método apagaria quer os círculos pensados quer as linhas entre as posições anteriores. Posto isto, para tentar contornar a situação, pensámos em colocar dois *panels* sobrepostos, com o objetivo de se visualizar, nos dois, os desenhos. Num desenhar-se-ia a trajetória, no outro, os corpos. Assim, o método *Clear*() para apagar os corpos já não afetaria a trajetória, pois esse método estaria aplicado apenas ao *panel* dos corpos. Contudo, esta ideia também não teve sucesso, pois não encontrámos maneira de, nem com a propriedade *BackColor.Transparent*, nem com os "*Bring to Front*"/"*Bring to Back*", conseguir visualizar os desenhos nos *panels*, simultaneamente. Finalmente, a ideia que teve sucesso, partiu do professor orientador. Corresponde à criação de uma lista que irá guardar todos os valores das posições dos corpos e, com estes, o desenho de múltiplas linhas que irá ligar os pontos das últimas posições. Como o desenho se vai repetir a cada *tick*, isto é, todas as linhas que ligam os pontos das posições anteriores vão ser redesenhadas a cada *tick* do *timer*, então, o método Clear(), apesar de continuar a apagar os corpos e as linhas, não vai afetar o desenho da trajetória. Quando procedemos a testes apercebemo-nos que não estava a ser desenhado qualquer trajetória. Após aprofundarmos o problema, e com a ajuda do professor, concluímos que se tratava de um problema de referências. Ao contrário do que se pensava, os valores da lista estavam a ser alterados, não um a um, mas sim todos, simultaneamente, daí não estar a ser visualizado qualquer linha. Acabámos por resolver este problema ao criar dois métodos na

```
//Desenho
if (checkBoxTrajetória.Checked == true)
{
    //código da trajetória
    oA.DrawTraj(g);
    oB.DrawTraj(g);
}

//desenho do movimento
if (checkBoxVetores.Checked == true)
{
    //Desenho com vetores
    oA.DrawV(g);
    oB.DrawV(g);
}
else
{
    //desenho sem vetores
    oA.DrawSV(g);
    oB.DrawSV(g);
```

classe "Vector2D", que eram necessários para gerar um valor novo a cada *tick* na lista dos pontos da posição.

5.2. Problemas Físicos

O problema no cerne desta aplicação é a física tangente ao movimento que tínhamos como propósito simular e como a aplicar na programação. Apesar de ser aparentemente impossível simular o movimento gravítico apenas com as equações clássicas da cinemática (equações horárias do movimento), conseguimos atingir esse objetivo:

- Problema do movimento dos corpos: Numa primeira fase, sabíamos que o código para o movimento teria de estar no evento *tick* do *timer*. Decidimos, também, para organizar o código, criar um método chamado Movimento() na classe "Corpo". Procedemos, depois, para o estudo físico do problema. Através da expressão da Lei Universal da Gravidade e da 3ª Lei de Newton, deduzimos uma equação para a componente escalar da aceleração dos corpos:

$$F = G \times \frac{M_A \times M_B}{r^2}$$

$$\vec{F} = m \times \vec{a}$$

Legenda:

G - constante gravitacional universal;

r - distância entre os dois corpos;

O que equivale a:

Componente escalar da aceleração do corpo A: $a_A = \frac{G \times m_B}{r^2 \times m_A}$

Componente escalar da aceleração do corpo B: $a_B = \frac{G \times m_A}{r^2 \times m_B}$

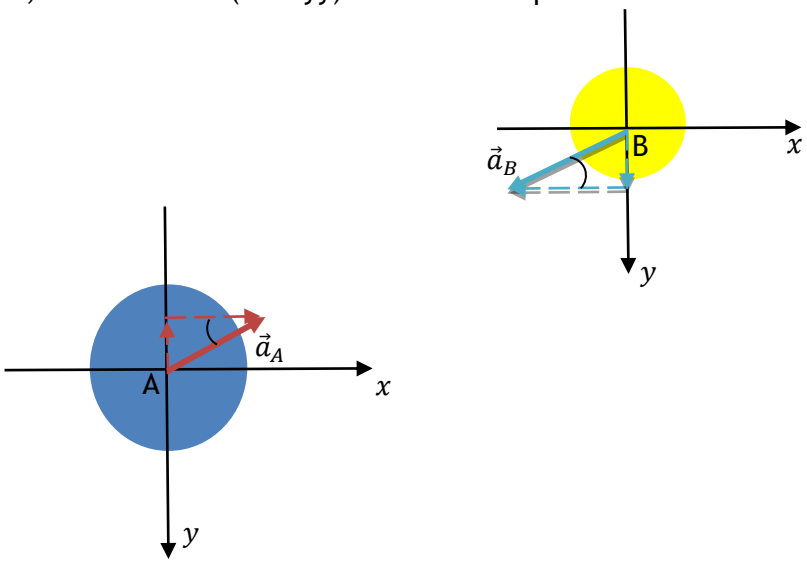
Posto isto, e visto que também é uma propriedade do vetor, criámos um objeto chamado "escalar" na classe "Vector2D". Sabendo as equações clássicas do movimento sabe-se que:

$$\vec{X}_{t+1} = \vec{X}_t + \vec{V}_{t+1} \times \Delta t$$

E que:

$$\vec{V}_{t+1} = \vec{V_t} + \vec{a} \times t$$

Por isso, como o grande objetivo é deduzir uma expressão para a próxima posição (no próximo *tick*), concluímos que o próximo passo seria obter uma maneira de deduzir o valor do vetor aceleração, em cada eixo (xx e yy). Partindo do que sabemos:



Assim, partindo de conceitos trigonométricos:

$$a_{A_x} = a_A \times \cos\alpha \Leftrightarrow a_{A_x} = \frac{x_B - x_A}{r} \times a_A \quad | \quad a_{B_x} = a_B \times \cos\alpha \Leftrightarrow a_{B_x} = \frac{x_A - x_B}{r} \times a_B$$

$$a_{A_y} = a_A \times \sin\alpha \Leftrightarrow a_{A_y} = \frac{y_B - y_A}{r} \times a_A \quad | \quad a_{B_y} = a_B \times \sin\alpha \Leftrightarrow a_{B_y} = \frac{y_A - y_B}{r} \times a_B$$

Feitas as deduções físicas, passámos à parte da programação. Na classe "Corpo" criámos, como já referimos, um método chamado Movimento(). Nesse método, programámos as equações que eram comuns aos dois corpos, isto é, a da posição (*SpacialLocation*) e a da velocidade, separando, respetivamente, pelos dois eixos:

```
public void Movimento(double t, Vector2D Acceleration)
{
    //código do movimento
    double elapsed = t / 1000;
    Velocity.X += Acceleration.X * elapsed;
    Velocity.Y += Acceleration.Y * elapsed;
    SpacialLocation.X += Velocity.X * elapsed;
    SpacialLocation.Y += Velocity.Y * elapsed;
}
```

Nota: Como se trata de um *timer* e de uma animação, que é atualizada a cada *tick* desse timer, $\Delta t = 1$, o que corresponde ao que chamámos "elapsed".

E no código do *Form* programámos as equações das grandezas que dependiam do corpo - a força gravítica e a aceleração:

```
//constantes
double G = 6.67E-11;
double r = Math.Sqrt(Math.Pow((oB.SpacialLocation.X - oA.SpacialLocation.X), 2) +
    Math.Pow((oB.SpacialLocation.Y - oA.SpacialLocation.Y), 2));
double F = G * ((oB.Mass * oA.Mass) / Math.Pow(r, 2));
//
//
//componentes escalares dos vetores aceleração
oA.Acceleration.escalar = F / oA.Mass;
oB.Acceleration.escalar = F / oB.Mass;
//

//valores das componentes da aceleração dos corpos

//corpo A
oA.Acceleration.X = ((oB.SpacialLocation.X - oA.SpacialLocation.X) / r) * oA.Acceleration.escalar;
oA.Acceleration.Y = ((oB.SpacialLocation.Y - oA.SpacialLocation.Y) / r) * oA.Acceleration.escalar;
//corpo B
oB.Acceleration.Y = ((oA.SpacialLocation.Y - oB.SpacialLocation.Y) / r) * oB.Acceleration.escalar;
oB.Acceleration.X = ((oA.SpacialLocation.X - oB.SpacialLocation.X) / r) * oB.Acceleration.escalar;
//
```

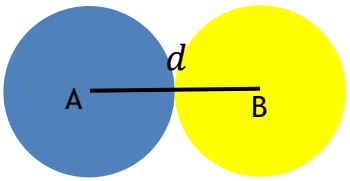
Posto isto, ao testarmos a aplicação com este código feito, reparámos que o corpo não estava a ser alterado pelo efeito da força, isto é, a aceleração mantinha-se constante e os corpos não estavam a sofrer o efeito um do outro. Ao aprofundarmos o problema, com a ajuda do professor, concluímos que ao utilizarmos massas com valores minúsculos, embora o efeito estivesse presente e a aceleração estivesse a mudar, essa alteração era extremamente reduzida (≈ na ordem dos 10^{-10}), ao ponto de não conseguirmos visualizar nenhuma reação.

Por conseguinte, de modo a resolver este percalço, criámos uma variável à qual chamámos "*MassScale*". Atribuímos o valor 10^{16} (depois de testarmos vários) e utilizámos o produto entre a *MassScale* e o valor que o utilizador colocaria para calcular a aceleração e todas as restantes grandezas.

```
// operational parameters
const double MassScale = 1E16;
```

```
oA.Mass = Convert.ToDouble(textBoxMassaA.Text) * MassScale;
oB.Mass = Convert.ToDouble(textBoxMassaB.Text) * MassScale;
```

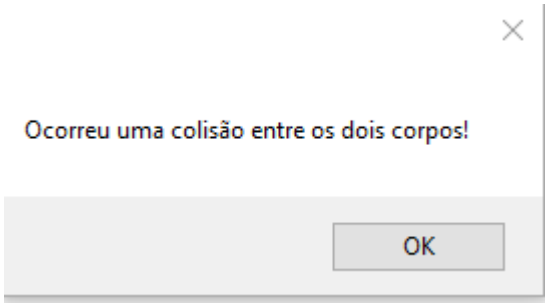
- Problema da colisão entre os corpos: Ao testarmos a aplicação deparámo-nos com situações em que os corpos colidiam. Para resolver esse problema:



Concluímos que, quando há colisão,

$$d \leq r_A + r_B$$

Posto isto, para solucionar o problema, no que toca à programação, criámos uma condição em que se $d \leq r_A + r_B$, o tempo parava e uma *MessageBox* era exibida ao utilizador:




```
//colisão
if (r < oA.DrawingRadius + oB.DrawingRadius)
{
    timer1.Stop();
    MessageBox.Show("Ocorreu uma colisão entre os dois corpos!");
}
//
```

6. Conclusões

Neste trabalho abordámos o assunto "Será que é possível desenvolver uma aplicação que simule o movimento causado pela força gravítica entre dois corpos celestes, apenas com equações (clássicas) do movimento?", e concluímos que não só conseguimos, como também superámos as expetativas que tínhamos depositado neste projeto.

De todos os objetivos a que nos propusemos inicialmente a cumprir, somente alguns não foram realizados, uma vez que o tempo foi escasso, e para que pudéssemos terminar e melhorar certos aspetos (mais relevante na nossa perspetiva), tivemos que abdicar de outros. Alguns deles passaram pela eliminação da *trackbar* da velocidade e da grelha quadriculada no *panel*. Além disso, ainda há certos *bugs* e imperfeições para limar, tais como o problema do *flickering* que está no movimento dos corpos e que não conseguimos resolver a tempo.

7. Reflexão Final

Gostámos muito de desenvolver este simulador. Considerámos este período de trabalho muito importante para o nosso conhecimento, compreensão e aprofundamento da unidade de programação, visto que nos permitiu explorar ao mesmo tempo uma área tão imprescindível nos dias que correm como a programação e uma área pela qual temos tango gosto como a física, sendo-nos uma grande aliada nas profissões que pretendemos seguir.

O objetivo atingindo foi, aliás, melhor que o previsto. Contudo, ainda queremos melhorar alguns pontos, principalmente o problema do *flickering*, pois ambos concordamos que torna a animação menos suave e agradável para o utilizador. Além disso, gostávamos, se possível, alongar o sistema para mais de dois corpos em movimento.

8. Referências

1. https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx

2. http://stackoverflow.com/

Ambos acedidos ao longo do desenvolvimento do projeto

9. Anexos

9.1. Código-fonte

Código da classe criada "Corpo":

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


namespace WindowsFormsApplication22
{

    class AstronomicalObject
    {

        // Physical properties
        public double Mass { get; set; }
        public Vector2D SpacialLocation { get; set; }
        public Vector2D Velocity { get; set; }
        public Vector2D Acceleration { get; set; }
        public Brush Brush { get; set; }
        public int DrawingRadius { get; set; }
        public List<Vector2D> Trajectory = new List<Vector2D>();

        //Desenhos
        public void DrawV(Graphics g)
        {
            // Desenho do corpo
            g.FillEllipse(Brush,
                (int)SpacialLocation.X - DrawingRadius,
                (int)SpacialLocation.Y - DrawingRadius,
                2 * DrawingRadius,
                2 * DrawingRadius);
            //
            //Desenho do vetor velocidade
            Pen vectorV = new Pen(Color.ForestGreen, 4);
            vectorV.EndCap = LineCap.ArrowAnchor;
            Point Vi = new Point((int)SpacialLocation.X, (int)SpacialLocation.Y);
            Point Vf = new Point(Vi.X + ((int)Velocity.X) / 10, Vi.Y + ((int)Velocity.Y) /
10);
            g.DrawLine(vectorV, Vi, Vf);
            //
            //Desenho do vetor aceleração
            Pen vectorA = new Pen(Color.Red, 4);
            vectorA.EndCap = LineCap.ArrowAnchor;
            Point Ai = new Point((int)SpacialLocation.X, (int)SpacialLocation.Y);
            Point Af = new Point(Ai.X + ((int)Acceleration.X) / 100, Ai.Y +
((int)Acceleration.Y) / 100);
            g.DrawLine(vectorA, Ai, Af);
        }
```

```
        public void Movimento(double t, Vector2D Acceleration)
        {
            //código do movimento
            double elapsed = t / 1000;
            Velocity.X += Acceleration.X * elapsed;
            Velocity.Y += Acceleration.Y * elapsed;
            SpacialLocation.X += Velocity.X * elapsed;
            SpacialLocation.Y += Velocity.Y * elapsed;
        }

        public void DrawSV(Graphics g)
        {
            //desenho dos corpos
            g.FillEllipse(Brush,
               (int)SpacialLocation.X - DrawingRadius,
               (int)SpacialLocation.Y - DrawingRadius,
               2 * DrawingRadius,
               2 * DrawingRadius);
        }

        public void DrawTraj(Graphics g)
        {
            //condição inicial da trajetória
            if (Trajectory.Count < 2) return;
            //
            //código de desenho da trajetória
            Point[] points = new Point[Trajectory.Count];
            for (int i = 0; i < Trajectory.Count; i++)
            {
                points[i] = new Point((int)Trajectory[i].X, (int)Trajectory[i].Y);
            }
            Pen p = new Pen(Color.Black, 2);
            g.DrawLines(p, points);
        }

    }

}
```

Código da classe criada "Vector2D":

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WindowsFormsApplication22
{
    class Vector2D
    {public double X { get; set; }
        public double Y { get; set; }
        public double escalar { get; set; }

        public Vector2D() {
            X = 0;
            Y = 0;
        }

        public Vector2D(Vector2D v)
        {
            X = v.X;
            Y = v.Y;
        }
```

```
    }
}
```

Código do Form:

```
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.Drawing.Drawing2D;
using System.Threading;


namespace WindowsFormsApplication22
{
    public partial class Simulador : Form
    {
        //constante da lista da trajétoria
        private const int N = 10000;

        // operational parameters
        const double MassScale = 1E16;

        // Criar dois objetos da classe AstronomicalObject
        AstronomicalObject oA = new AstronomicalObject();
        AstronomicalObject oB = new AstronomicalObject();

        //criar a variável do tempo
        double t = 0;

        ////variável da espessura da trajetória
        //int espessura = 1;

        public Simulador()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            //definição dos objetos a ser tratados
            oA.Brush = Brushes.Pink;
            oB.Brush = Brushes.LightBlue;
            oA.SpacialLocation = new Vector2D();
            oB.SpacialLocation = new Vector2D();
            oA.Velocity = new Vector2D();
            oB.Velocity = new Vector2D();
            oA.Acceleration = new Vector2D();
            oB.Acceleration = new Vector2D();
            //
            // Inicia-los com os valores preenchidos pelo utilizador
            oA.DrawingRadius = (int)Convert.ToDouble(textBoxMassaA.Text) / 10;
            oB.DrawingRadius = (int)Convert.ToDouble(textBoxMassaB.Text) / 10;
            oA.Mass = Convert.ToDouble(textBoxMassaA.Text) * MassScale;
            oB.Mass = Convert.ToDouble(textBoxMassaB.Text) * MassScale;
            oA.SpacialLocation.X = Convert.ToDouble(textBoxXoA.Text);
            oB.SpacialLocation.X = Convert.ToDouble(textBoxXoB.Text);
            oA.SpacialLocation.Y = Convert.ToDouble(textBoxYoA.Text);
            oB.SpacialLocation.Y = Convert.ToDouble(textBoxYoB.Text);
            oA.Velocity.X = Convert.ToDouble(textBoxVoxA.Text);
```

```
            oB.Velocity.X = Convert.ToDouble(textBoxVoxB.Text);
            oA.Velocity.Y = Convert.ToDouble(textBoxVoyA.Text);
            oB.Velocity.Y = Convert.ToDouble(textBoxVoyB.Text);
        }

        private void buttonIniciar_Click(object sender, EventArgs e)
        {
            //definição dos objetos a ser tratados
            oA.Brush = Brushes.Pink;
            oB.Brush = Brushes.LightBlue;
            oA.SpacialLocation = new Vector2D();
            oB.SpacialLocation = new Vector2D();
            oA.Velocity = new Vector2D();
            oB.Velocity = new Vector2D();
            oA.Acceleration = new Vector2D();
            oB.Acceleration = new Vector2D();
            //
            // Inicia-los com os valores preenchidos pelo utilizador
            oA.DrawingRadius = (int)Convert.ToDouble(textBoxMassaA.Text) / 10;
            oB.DrawingRadius = (int)Convert.ToDouble(textBoxMassaB.Text) / 10;
            oA.Mass = Convert.ToDouble(textBoxMassaA.Text) * MassScale;
            oB.Mass = Convert.ToDouble(textBoxMassaB.Text) * MassScale;
            oA.SpacialLocation.X = Convert.ToDouble(textBoxXoA.Text);
            oB.SpacialLocation.X = Convert.ToDouble(textBoxXoB.Text);
            oA.SpacialLocation.Y = Convert.ToDouble(textBoxYoA.Text);
            oB.SpacialLocation.Y = Convert.ToDouble(textBoxYoB.Text);
            oA.Velocity.X = Convert.ToDouble(textBoxVoxA.Text);
            oB.Velocity.X = Convert.ToDouble(textBoxVoxB.Text);
            oA.Velocity.Y = Convert.ToDouble(textBoxVoyA.Text);
            oB.Velocity.Y = Convert.ToDouble(textBoxVoyB.Text);
            //criação do contexto Gráfico do Panel
            Graphics g = panelSimulação.CreateGraphics();
            //limpar o panel
            g.Clear(panelSimulação.BackColor);
            //
            //Clear da Trajetória
            oA.Trajectory.Clear();
            oB.Trajectory.Clear();
            //
            // Desenhar os objetos
            oA.DrawSV(g);
            oB.DrawSV(g);
            ////DISABLE DAS TEXTBOXES
            textBoxMassaA.Enabled = false;
            textBoxMassaB.Enabled = false;
            textBoxXoA.Enabled = false;
            textBoxYoA.Enabled = false;
            textBoxXoB.Enabled = false;
            textBoxYoB.Enabled = false;
            textBoxVoxA.Enabled = false;
            textBoxVoyA.Enabled = false;
            textBoxVoxB.Enabled = false;
            textBoxVoyB.Enabled = false;
            //
            //inicio do tempo
            timer1.Start();
        }

        private void buttonParar_Click(object sender, EventArgs e)
        {
            //Parar o tempo
            timer1.Stop();
            //
            //ENABLED DAS TEXTBOXES
            textBoxMassaA.Enabled = true;
```

```
            textBoxMassaB.Enabled = true;
            textBoxXoA.Enabled = true;
            textBoxYoA.Enabled = true;
            textBoxXoB.Enabled = true;
            textBoxYoB.Enabled = true;
            textBoxVoxA.Enabled = true;
            textBoxVoyA.Enabled = true;
            textBoxVoxB.Enabled = true;
            textBoxVoyB.Enabled = true;
        }

        private void timer1_Tick(object sender, EventArgs e)
        {

            //definição do graphics no panelsimulação
            Graphics g = panelSimulação.CreateGraphics();
            //
            //limpar o panel
            g.Clear(panelSimulação.BackColor);
            //
            //colocar na label do tempo, o cronómetro
            this.t += timer1.Interval;
            labelT.Text = (Convert.ToString(this.t / 1000) + "s");
            //
            //constantes
            double G = 6.67E-11;
            double r = Math.Sqrt(Math.Pow((oB.SpacialLocation.X - oA.SpacialLocation.X), 2)
+
                Math.Pow((oB.SpacialLocation.Y - oA.SpacialLocation.Y), 2));
            double F = G * ((oB.Mass * oA.Mass) / Math.Pow(r, 2));
            //
            //colisão
            if (r < oA.DrawingRadius + oB.DrawingRadius)
            {
                timer1.Stop();
                MessageBox.Show("Ocorreu uma colisão entre os dois corpos!");
            }
            //
            //componentes escalares dos vetores aceleração
            oA.Acceleration.escalar = F / oA.Mass;
            oB.Acceleration.escalar = F / oB.Mass;
            //

            //valores das componentes da aceleração dos corpos

            //corpo A
            oA.Acceleration.X = ((oB.SpacialLocation.X - oA.SpacialLocation.X) / r) *
oA.Acceleration.escalar;
            oA.Acceleration.Y = ((oB.SpacialLocation.Y - oA.SpacialLocation.Y) / r) *
oA.Acceleration.escalar;
            //corpo B
            oB.Acceleration.Y = ((oA.SpacialLocation.Y - oB.SpacialLocation.Y) / r) *
oB.Acceleration.escalar;
            oB.Acceleration.X = ((oA.SpacialLocation.X - oB.SpacialLocation.X) / r) *
oB.Acceleration.escalar;
            //
            //Condições para a Lista da Trajetória
            if (oA.Trajectory.Count > N)
            {
                oA.Trajectory.RemoveAt(0);
                oB.Trajectory.RemoveAt(0);
            }
            oA.Trajectory.Add(new Vector2D(oA.SpacialLocation));
            oB.Trajectory.Add(new Vector2D(oB.SpacialLocation));
            //
```

```
            //movimento
            oA.Movimento(timer1.Interval, oA.Acceleration);
            oB.Movimento(timer1.Interval, oB.Acceleration);
            //
            //Desenho
            if (checkBoxTrajetória.Checked == true)
            {
                //código da trajetória
                oA.DrawTraj(g);
                oB.DrawTraj(g);
            }

            //desenho do movimento
            if (checkBoxVetores.Checked == true)
            {
                //Desenho com vetores
                oA.DrawV(g);

                  oB.DrawV(g);
            }
            else
            {
                //desenho sem vetores
                oA.DrawSV(g);
                oB.DrawSV(g);
            }
            //Thread.Sleep(50);
        }
        private void buttonContinuar_Click(object sender, EventArgs e)
        {
            //Reinício do tempo
            timer1.Start();
        }

        private void checkBoxTrajetória_CheckedChanged(object sender, EventArgs e)
        {
            //contexto gráfico do panel
            Graphics g = panelSimulação.CreateGraphics();
            g.Clear(panelSimulação.BackColor);
            //
            if (checkBoxTrajetória.Checked == true)
            {
                //código da trajetória
                oA.DrawTraj(g);
                oB.DrawTraj(g);
            }
            if (checkBoxVetores.Checked == true)
            {
                oA.DrawV(g);
                oB.DrawV(g);

            }
            else
            {
                oA.DrawSV(g);
                oB.DrawSV(g);
            }
        }
        private void checkBoxVetores_CheckedChanged(object sender, EventArgs e)
        {
            Graphics g = panelSimulação.CreateGraphics();
            g.Clear(panelSimulação.BackColor);
            //
            if (checkBoxTrajetória.Checked == true)
            {
```

```
                //código da trajetória
                oA.DrawTraj(g);
                oB.DrawTraj(g);
            }
            if (checkBoxVetores.Checked == true)
            {
                oA.DrawV(g);
                oB.DrawV(g);

            }
            else
            {
                oA.DrawSV(g);
                oB.DrawSV(g);
            }
        }
private void textBoxMassaA_TextChanged(object sender, EventArgs e)
        {
            //código de proteção da textbox
            int n;
            bool isNumeric = int.TryParse(textBoxMassaA.Text, out n);
            if (isNumeric == false)
            {
                MessageBox.Show("Insira Valor Numérico");
                if (textBoxMassaA.Text.Length == 0)
                {
                    buttonIniciar.Enabled = false;
                    buttonContinuar.Enabled = false;
                    buttonPausa.Enabled = false;
                    return;
                }
                //Apagar o último caracter
                textBoxMassaA.Text = textBoxMassaA.Text.Substring(0,
                    textBoxMassaA.Text.Length - 1);

            }
            else
            {
                buttonIniciar.Enabled = true;
                buttonContinuar.Enabled = true;
                buttonPausa.Enabled = true;
            }


        }

        private void textBoxMassaB_TextChanged(object sender, EventArgs e)
        {
            //código de proteção da textbox
            int n;
            bool isNumeric = int.TryParse(textBoxMassaB.Text, out n);
            if (isNumeric == false)
            {
                MessageBox.Show("Insira Valor Numérico");
                if (textBoxMassaB.Text.Length == 0)
                {
                    buttonIniciar.Enabled = false;
                    buttonContinuar.Enabled = false;
                    buttonPausa.Enabled = false;
                    return;
                }
                //Apagar o último caracter
                textBoxMassaB.Text = textBoxMassaB.Text.Substring(0,
                    textBoxMassaB.Text.Length - 1);
```

```
        }
        else
        {
            buttonIniciar.Enabled = true;
            buttonContinuar.Enabled = true;
            buttonPausa.Enabled = true;
        }
    }

    private void textBoxXoA_TextChanged(object sender, EventArgs e)
    {
        //código de proteção da textbox
        int n;
        bool isNumeric = int.TryParse(textBoxXoA.Text, out n);
        if (isNumeric == false)
        {
            MessageBox.Show("Insira Valor Numérico");
            if (textBoxXoA.Text.Length == 0)
            {
                buttonIniciar.Enabled = false;
                buttonContinuar.Enabled = false;
                buttonPausa.Enabled = false;
                return;
            }
            //Apagar o último caracter
            textBoxXoA.Text = textBoxXoA.Text.Substring(0,
                textBoxXoA.Text.Length - 1);

        }
        else
        {
            buttonIniciar.Enabled = true;
            buttonContinuar.Enabled = true;
            buttonPausa.Enabled = true;
        }
    }

    private void textBoxXoB_TextChanged(object sender, EventArgs e)
    {
        //código de proteção da textbox
        int n;
        bool isNumeric = int.TryParse(textBoxXoB.Text, out n);
        if (isNumeric == false)
        {
            MessageBox.Show("Insira Valor Numérico");
            if (textBoxXoB.Text.Length == 0)
            {
                buttonIniciar.Enabled = false;
                buttonContinuar.Enabled = false;
                buttonPausa.Enabled = false;
                return;
            }
            //Apagar o último caracter
            textBoxXoB.Text = textBoxXoB.Text.Substring(0,
                textBoxXoB.Text.Length - 1);

        }
        else
        {
            buttonIniciar.Enabled = true;
            buttonContinuar.Enabled = true;
            buttonPausa.Enabled = true;
        }
    }
```

```
        private void textBoxYoA_TextChanged(object sender, EventArgs e)
        {
            //código de proteção da textbox
            int n;
            bool isNumeric = int.TryParse(textBoxYoA.Text, out n);
            if (isNumeric == false)
            {
                MessageBox.Show("Insira Valor Numérico");
                if (textBoxYoA.Text.Length == 0)
                {
                    buttonIniciar.Enabled = false;
                    buttonContinuar.Enabled = false;
                    buttonPausa.Enabled = false;
                    return;
                }
                //Apagar o último caracter
                textBoxYoA.Text = textBoxYoA.Text.Substring(0,
                    textBoxYoA.Text.Length - 1);

            }
            else
            {
                buttonIniciar.Enabled = true;
                buttonContinuar.Enabled = true;
                buttonPausa.Enabled = true;
            }
        }

        private void textBoxYoB_TextChanged(object sender, EventArgs e)
        {
            //código de proteção da textbox
            int n;
            bool isNumeric = int.TryParse(textBoxYoB.Text, out n);
            if (isNumeric == false)
            {
                MessageBox.Show("Insira Valor Numérico");
                if (textBoxYoB.Text.Length == 0)
                {
                    buttonIniciar.Enabled = false;
                    buttonContinuar.Enabled = false;
                    buttonPausa.Enabled = false;
                    return;
                }
                //Apagar o último caracter
                textBoxYoB.Text = textBoxYoB.Text.Substring(0,
                    textBoxYoB.Text.Length - 1);

            }
            else
            {
                buttonIniciar.Enabled = true;
                buttonContinuar.Enabled = true;
                buttonPausa.Enabled = true;
            }
        }

        private void textBoxVoxA_TextChanged(object sender, EventArgs e)
        {
            //código de proteção da textbox
            int n;
            bool isNumeric = int.TryParse(textBoxVoxA.Text, out n);
            if (isNumeric == false)
            {
                MessageBox.Show("Insira Valor Numérico");
                if (textBoxVoxA.Text.Length == 0)
```

```
            {
                buttonIniciar.Enabled = false;
                buttonContinuar.Enabled = false;
                buttonPausa.Enabled = false;
                return;
            }
            //Apagar o último caracter
            textBoxVoxA.Text = textBoxVoxA.Text.Substring(0,
                textBoxVoxA.Text.Length - 1);

        }
        else
        {
            buttonIniciar.Enabled = true;
            buttonContinuar.Enabled = true;
            buttonPausa.Enabled = true;
        }
    }



    private void textBoxVoxB_TextChanged(object sender, EventArgs e)
    {
        //código de proteção da textbox
        int n;
        bool isNumeric = int.TryParse(textBoxVoxB.Text, out n);
        if (isNumeric == false)
        {
            MessageBox.Show("Insira Valor Numérico");
            if (textBoxVoxB.Text.Length == 0)
            {
                buttonIniciar.Enabled = false;
                buttonContinuar.Enabled = false;
                buttonPausa.Enabled = false;
                return;
            }
            //Apagar o último caracter
            textBoxVoxB.Text = textBoxVoxB.Text.Substring(0,
                textBoxVoxB.Text.Length - 1);

        }
        else
        {
            buttonIniciar.Enabled = true;
            buttonContinuar.Enabled = true;
            buttonPausa.Enabled = true;
        }
    }

    private void textBoxVoyA_TextChanged(object sender, EventArgs e)
    {
        //código de proteção da textbox
        int n;
        bool isNumeric = int.TryParse(textBoxVoyA.Text, out n);
        if (isNumeric == false)
        {
            MessageBox.Show("Insira Valor Numérico");
            if (textBoxVoyA.Text.Length == 0)
            {
                buttonIniciar.Enabled = false;
                buttonContinuar.Enabled = false;
                buttonPausa.Enabled = false;
                return;
            }
            //Apagar o último caracter
```

```
                textBoxVoyA.Text = textBoxVoyA.Text.Substring(0,
                    textBoxVoyA.Text.Length - 1);

            }
            else
            {
                buttonIniciar.Enabled = true;
                buttonContinuar.Enabled = true;
                buttonPausa.Enabled = true;
            }
        }

        private void textBoxVoyB_TextChanged(object sender, EventArgs e)
        {
            //código de proteção da textbox
            int n;
            bool isNumeric = int.TryParse(textBoxVoyB.Text, out n);
            if (isNumeric == false)
            {
                MessageBox.Show("Insira Valor Numérico");
                if (textBoxVoyB.Text.Length == 0)
                {
                    buttonIniciar.Enabled = false;
                    buttonContinuar.Enabled = false;
                    buttonPausa.Enabled = false;
                    return;
                }
                //Apagar o último caracter
                textBoxVoyB.Text = textBoxVoyB.Text.Substring(0,
                    textBoxVoyB.Text.Length - 1);

            }
            else
            {
                buttonIniciar.Enabled = true;
                buttonContinuar.Enabled = true;
                buttonPausa.Enabled = true;
            }
        }

    }

}
```

9.2. Manual do utilizador

1. Insira valores numéricos na tabela para cada um dos corpos, de modo a definir a massa, a posição inicial e a velocidade inicial de cada um;
2. Selecione as *checkboxes* (“Mostrar Vetores” ou “Mostrar Trajetória”) de acordo com a sua preferência, podendo serem desligadas a qualquer instante;
3. Selecione o botão “INICIAR” para começar o movimento dos corpos;
4. Por fim, tem à disposição o botão “PAUSA” sempre que quiser parar o movimento, e o botão “CONTINUAR” para o prosseguir. Pode ainda selecionar o botão “INICIAR” se desejar reiniciar todo o movimento, voltando o tempo ao instante zero;