****

Ambientes de programação gráfica

RELATÓRIO DE PROJETO

Sistema de Partículas

Docente:

Luís Romero

Diogo miguel almeida amorim Nº:18463

tiago AnDré Campos castro Nº:18459

## Introdução

Através da lista submetida pelo docente, foi-nos atribuído o projeto “Sistemas de partículas” no qual iremos abordar o desenvolvimento do mesmo neste relatório abordando os métodos e classes que foram utilizados para conseguir realizar este mesmo trabalho.

O nosso trabalho deveria atingir os seguintes objetivos:

* Implementar uma aplicação para simular uma dispersão de partículas no espaço.
* Cada vez que um utilizador prime o botão do rato na janela da aplicação são geradas n partículas (formas muito pequenas) que dispersam a partir do ponto onde foi premido o botão do rato.
* As partículas dispersam-se circularmente à volta do ponto, com uma velocidade inicial aleatória, e vão perdendo velocidade até pararem e desaparecerem.
* Caso colidam nas laterais da janela devem ricochetear.
* Representar o rato com uma mira e incluir um botão para terminar a aplicação.

O Sistema de Partículas, ou sistema de pontos materiais, designa-se um conjunto finito ou infinito de partículas, de tal modo que entre qualquer um dos seus pontos permanece invariável durante o movimento. Isto significa que apenas se irá considerar sistemas de párticas rígidos.

Assim apos estabelecer quais os nossos objetivos, utilizando os exemplo que o docente da cadeira nos foi dando e com alguns pesquisas tentamos criar na linguagem de programação C# , utilizando o Visual Estúdio, criar a nossa aplicação capaz de atingir os nosso objetivos.

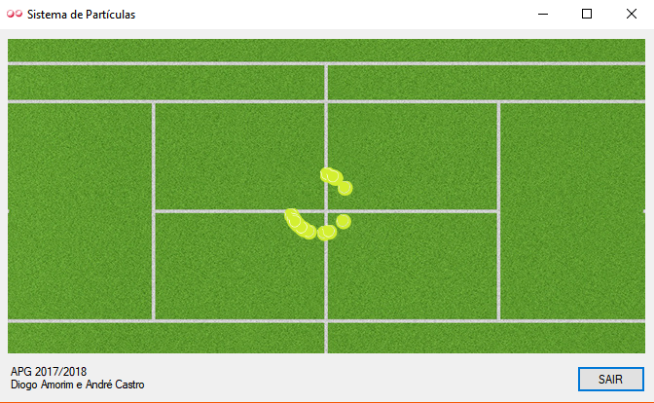


Figura 1-Programa Final com contexto de Ténis

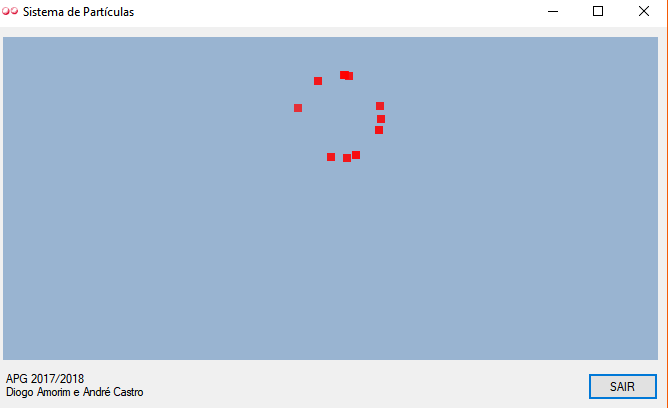


Figura 2- Programa Teste

## Desenvolvimento do Trabalho

Nestes tópicos vamos explorar os elementos chave da programação, ou seja, as classes e métodos com que fazem este programa gerir a “explosão de partículas”.

## Classe Conj Particulas

public List<Particula> Particulas;

public Conj\_Particulas()

{

Particulas = new List<Particula>();

}

//Esta classe apenas cria uma lista de Partículas onde irão ser guardadas todas as partículas que vão aparecer no ecrã durante a execução do programa.

## Classe Particula

TimeSpan TempoVida = new TimeSpan(0, 0, 0, rg.Next(2, 8));

//É criada uma variável TempoVida do tipo TimeSpan onde irá guardar o tempo de vida, em segundos, das partículas que aparecerão no ecrã. O Tempo de Vida das partículas está a ser gerado aleatoriamente para cada partícula com valores entre 2 e 8 segundos.

DateTime? DataNasc;

//DataNasc é uma variável do tipo DataTime onde está a ser guardado a data exata a que a partícula nasce para ser possível calcular quando é que a partícula deve desaparecer.

private PointF \_Location;

public PointF Location { get { return \_Location; } set { \_Location = value; } }

private PointF \_Velocity;

public PointF Velocity { get { return \_Velocity; } set { \_Velocity = value; } }

//São criadas variáveis do tipo PointF onde guardam a posição exata onde a partícula está e a Velocidade e direção da mesma.

public static Random rg = new Random();

//Variavel do tipo Random para gerar valores aleatórios ao longo da classe.

public static PointF RandomVector(float speed)

{

float rangle = (float)((rg.NextDouble() \* (Math.PI \* 2)));

//rangle é uma variável do tipo float que vai gerar um ángulo aleatório para ser disparada a partícula. Para isso multiplicamos um valor aleatório entre 0 e 1 por 2π (equivale a uma circunferência, 360º).

speed \*= (float)rg.NextDouble();

//Recebendo a velocidade como método da função, a esta será multiplicado um valor entre 0 e 1, fazendo assim com que a velocidade vá diminuindo ao longo do tempo.

//Esta função vai retornar um vetor com a nova posição da partícula de acordo com o ângulo gerado anteriormente multiplicado pela velocidade da partícula, e assim saber as coordenadas da partícula.

return new PointF((float)Math.Cos(rangle) \* speed, (float)Math.Sin(rangle) \* speed);

}

**// CONSTRUTORES**

public Particula(PointF Location, float Speed) : this(Location, RandomVector(Speed))

{

}

public Particula(PointF Location, PointF Velocity)

{

\_Location = Location;

\_Velocity = Velocity;

//Define a localização e a Velocidade/Direção da partícula.

}

//Esta função irá ser executada enumeras vezes, para cada partícula no ecrã.

public bool PerformFrame(Conj\_Particulas part)

{

if (DataNasc == null) DataNasc = DateTime.Now;

//Verifica se a partícula tem uma DataNasc, e caso não tenha sido atribuída, é lhe dada a data atual.

\_Location = new PointF(\_Location.X + \_Velocity.X, \_Location.Y + \_Velocity.Y);

//A localização da partícula será atualizada, somando na posição do X, a localização atual com a velocidade e no Y, a localização de Y com a respetiva velocidade.

\_Velocity = new PointF(\_Velocity.X \* 0.99f, \_Velocity.Y \* 0.99f);

//A velocidade será alterada, sendo multiplicada a velocidade atual por 0.99, fazendo com que a partícula perca a velocidade ao longo do tempo.

if (\_Location.X > 640 || \_Location.X < 0) \_Velocity.X = -Velocity.X;

if (\_Location.Y > 310 || \_Location.Y < 0) \_Velocity.Y = -Velocity.Y;

//Estas duas condições fazem com que a partícula, caso ultrapasse as bordas da pictureBox, a sua velocidade passe a ser a inversa, fazendo com que ao bater na pictureBox, voltem para trás, fazendo assim que haja um ricochete.

return DateTime.Now - DataNasc > TempoVida;

//A função vai retornar true se a partícula ultrapassou o seu tempo de vida e false se ainda permanece viva.

}

public void Draw(Graphics g)

{

Image image;

image = TrabalhoAPG.Properties.Resources.tennis\_ball;

//Nesta função é carregada uma imagem de uma bola de tênis que representará as partículas no programa, e na função abaixo irão ser desenhadas as bolas na pictureBox, de acordo com a localização de X e Y do clique e com o tamanho de 16 por 16.

g.DrawImage(image, Location.X, Location.Y,16,16);

}

## Form1

private Thread GameThread;

private Conj\_Particulas cpart = new Conj\_Particulas();

//Aqui é definido um Thread e é criada uma variável do tipo cpart, que nada mais é que uma lista de partículas, ou conjunto de partículas como o próprio nome da classe indica.

public void Proced()

{

//Aqui é onde grande parte do programa acontece, onde são movidas as partículas a cada 50 Milisegundos, e são removidas caso o seu tempo de vida esteja ultrapassado.

try

{

for (; ; )

{

Thread.Sleep(50);

List<Particula> removeParticulas = new List<Particula>();

//Aqui é criada uma lista de Particulas onde irá conter as partículas removidas

lock (cpart.Particulas)

{

foreach (Particula p in cpart.Particulas)

{

if (p.PerformFrame(cpart))

{

removeParticulas.Add(p);

//Para cada Particula dentro do conjunto das Particulas existentes, será executada a função PerformFrame que, como vimos anteriormente irá calcular as novas coordenadas da partícula e por fim vê se está viva. Caso a partícula em questão esteja morta, será removida aqui, adicionando-a á lista das partículas removidas e posteriormente retirada da lista das Particulas.

}

}

}

foreach (var part\_remove in removeParticulas)

{

cpart.Particulas.Remove(part\_remove);

}

Invoke((MethodInvoker)(() =>

{

pictureBox1.Invalidate();

pictureBox1.Update();

}));

//Aqui em cima, a pictureBox irá ser atualizada, fazendo com que crie a ilusão das partículas se mexerem.

}

}

catch {

}

}

//Aqui em baixo está o construtor do form, onde para além da normal funcionalidade, irá definir o cursor para a mira.

public Form1()

{

InitializeComponent();

pictureBox1.Cursor = new Cursor(GetType(), "scope.cur");

}

public void Form1\_Load(object sender, EventArgs e)

{

GameThread = new Thread(Proced);

GameThread.Start();

//No load do form vai ser criado um novo Thread para a função Proced e também executar o Thread.

}

private void pictureBox1\_Paint(object sender, PaintEventArgs e)

{

foreach (Particula p in cpart.Particulas)

{

p.Draw(e.Graphics);

}

//Para cada partícula existente, irá executar a função que irá desenhar a partícula no ecrã.

}

private void pictureBox1\_MouseClick(object sender, MouseEventArgs e)

{

PointF clicklocation = new PointF(e.X,e.Y);

//Guarda a posição do clique do rato.

Random rnd1 = new Random();

int NrParticulas = rnd1.Next(10, 30);

//Gera um numero aleatório de partículas, entre 10 e 30.

lock (cpart.Particulas)

{

For (int i = 0; i < NrParticulas; i++)

{

Particula createparticle = new Particula(clicklocation, rnd1.Next(1, 5));

//Aqui irá criar todas as partículas por clique, sendo que a velocidade vai variar entre 1 e 5. Posteriormente são adicionadas á lista de Particulas.

cpart.Particulas.Add(createparticle);

}

}

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

GameThread.Abort();

GameThread = null;

//Ao fechar a aplicação, irá fechar o Thread criado para o jogo.

}

private void button1\_Click(object sender, EventArgs e)

{

Application.Exit();

//Esta função vai fechar o programa quando o utilizador clica no botão SAIR.

}

## Conclusão

Consegui-mos olhar para trás desde o inicio deste projeto e ver que tivemos várias dificuldades, mas consegui-mos supera-las fazendo com que alargasse-mos o nosso conhecimento e nos tornasse-mos mais “desenrascados” como programadores gerando soluções alternativas fazendo com que cresça-mos.

Dos objetivos que tivemos que alcançar os mais fáceis foram a criação do espaço em si, ou seja, o como e todas as suas variáveis.

O objetivo que provou ser de maior dificuldade a explosão em circulo com velocidades variadas o que exigia de nós programadores um conhecimento de ordens físicas.

Em síntese foi um trabalho pratico que apresentou desafio, e que nos ajudou a melhor as nossas técnicas em física e programação em c# pondo a prova as matérias que aprendemos durante o semestre e a nossa capacidade de adaptamento e resolução de problemas.