Relatório – 3º Fase

Carlos Passos - 16924

Tiago Oliveira - 16931

Index

CLASSE: “ClsTank” – Update

Nesta última fase fizemos alterações no nosso tank para podermos prosseguir com o trabalho e o acabar.

```
public enum Player
{
    P1 = 0, P2 = 1, CPU = 2
}
public Player player;
```

```
if (p == Player.P1)
{
    vel = 10;
    yaw = MathHelper.ToRadians(-45f);
    range = new Range(20, position);
    rangeToShoot = new Range(10, position);
}
else if (p == Player.CPU || p == Player.P2)
{
    vel = 5;
    aMax = 3;
    velMax = 8;
    yaw = MathHelper.ToRadians(90f);
    range = new Range(80, position);
    rangeToShoot = new Range(10, position);
    rangePursuit = new Range(15, position);
    rangeRun = new Range(5, position);
}
```

Cada tank tem um ID que o identifica (P1, P2 ou o CPU). Dependendo do ID, cada tank terá diferentes ranges, diferentes velocidades, acelerações e um yaw específico.

```
public void Update(KeyboardState kb, MouseState ms, GraphicsDevice gd, GameTime gt, MapGen mp)
{
    rotation = Matrix.Identity;
    Matrix scl = Matrix.CreateScale(scale);
    Vector2 center = new Vector2(gd.Viewport.Width / 2, gd.Viewport.Height / 2);
    Vector3 dir;
    Vector3 newPos = position;

    float delta_X = ms.X - center.X;
    float delta_Y = ms.Y - center.Y;

    if (player == Player.CPU || player == Player.P2)...

    if (player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.Up))...
    else if (player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.Down))...

    if (player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.Right))...
    else if (player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.Left))...


    if (MathHelper.ToDegrees(turrentAngle) < -90)...
    else if (MathHelper.ToDegrees(turrentAngle) > 90)...
    if (MathHelper.ToDegrees(cannonAngle) > 0)...
    else if (MathHelper.ToDegrees(cannonAngle) < -90)...
```

O cano do tank do P1 é controlado pelas setas e o do P2 pela direção do rato. O cano e a torre têm um limite de rotação sendo de [-90;90]

```
if (player == Player.P1)
{
    if ((player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.A)))
    {
        yaw += MathHelper.ToRadians(1f);
        wheels[1] += MathHelper.ToRadians(1);
    }
    if ((player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.D)))
    {
        yaw -= MathHelper.ToRadians(1f);
        wheels[1] += MathHelper.ToRadians(1);
    }

    direction = Vector3.Transform(Vector3.UnitX, Matrix.CreateFromYawPitchRoll(yaw, 0, 0));

    if ((player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.W)))
    {
        newPos = position + direction * vel * (float)gt.ElapsedGameTime.TotalSeconds;
        newPos.Y = mp.Heigth(newPos.X, newPos.Z);
        wheels[0] += 80 * MathHelper.ToRadians(2f) * (float)gt.ElapsedGameTime.TotalSeconds;
        move = Movement.FOLLOW;
    }

    if ((player == Player.P1 && Keyboard.GetState().IsKeyDown(Keys.S)))
    {
        newPos = position - direction * vel * (float)gt.ElapsedGameTime.TotalSeconds;
        newPos.Y = mp.Heigth(newPos.X, newPos.Z);
        wheels[0] += 80 * MathHelper.ToRadians(-2f) * (float)gt.ElapsedGameTime.TotalSeconds;
        move = Movement.RUN;
    }
}
```

Caso seja o player 1 , o Tank irá rodar no ‘A’ e ‘D’ movendo-se para a frente com o ‘W’ e para trás com o ‘S’.

```
else if (player == Player.P2)
{
    if ((player == Player.P2 && Keyboard.GetState().IsKeyDown(Keys.J)))
    {
        yaw += MathHelper.ToRadians(1f);
        wheels[1] += MathHelper.ToRadians(1);
    }
    if ((player == Player.P2 && Keyboard.GetState().IsKeyDown(Keys.L)))
    {
        yaw -= MathHelper.ToRadians(1f);
        wheels[1] += MathHelper.ToRadians(1);
    }

    direction = Vector3.Transform(Vector3.UnitX, Matrix.CreateFromYawPitchRoll(yaw, 0, 0));

    if ((player == Player.P2 && Keyboard.GetState().IsKeyDown(Keys.I)))
    {
        newPos = position + direction * vel * (float)gt.ElapsedGameTime.TotalSeconds;
        newPos.Y = mp.Heigth(newPos.X, newPos.Z);
        wheels[0] += 80 * MathHelper.ToRadians(2f) * (float)gt.ElapsedGameTime.TotalSeconds;
        move = Movement.FOLLOW;
    }

    if ((player == Player.P2 && Keyboard.GetState().IsKeyDown(Keys.K)))
    {
        newPos = position - direction * vel * (float)gt.ElapsedGameTime.TotalSeconds;
        newPos.Y = mp.Heigth(newPos.X, newPos.Z);
        wheels[0] += 80 * MathHelper.ToRadians(-2f) * (float)gt.ElapsedGameTime.TotalSeconds;
        move = Movement.RUN;
    }
}
```

Caso seja o player 2, o Tank tem o mesmo comportamento, porém com as letras ‘J’, ’L’, ’I’ e ’K’. A maneira como se calcula é, portanto, a mesma.

```
else if (player == Player.CPU)
{
    direction = Vector3.Transform(Vector3.UnitX, Matrix.CreateFromYawPitchRoll(yaw, 0, 0));

    if (range.OnRange(gm.tank1.range))
    {
        Vector3 directionSeek = gm.tank1.position - position;
        directionSeek.Normalize();
        Vector3 vseek = directionSeek * velMax;

        Vector3 v = direction * vel;
        a = (vseek - v);
        a.Normalize();
        a = a * aMax;

        v = v + a * (float)gt.ElapsedGameTime.TotalSeconds;

        direction = v;
        direction.Normalize();

        vel = v.Length();

        yaw = (float)Math.Atan2(-direction.Z, direction.X);

        if (move == Movement.RUN)
        {
            newPos = position - direction * vel * (float)gt.ElapsedGameTime.TotalSeconds;
            newPos.Y = mp.Heigth(newPos.X, newPos.Z);
        }
        else if (move == Movement.FOLLOW)
        {
            newPos = position + direction * vel * (float)gt.ElapsedGameTime.TotalSeconds;
            newPos.Y = mp.Heigth(newPos.X, newPos.Z);
        }
    }
    else
    {
        newPos = position + direction * vel * (float)gt.ElapsedGameTime.TotalSeconds;
        newPos.Y = mp.Heigth(newPos.X, newPos.Z);
    }
```

Para o CPU utilizou-se o que nós chamamos de “Boids”. Para isso começamos por calcular a direção que o tankCPU tem de fazer para seguir o tankP1. Para que não houvesse uma velocidade espontânea calculamos a velocidade desejada, depois a do momento e por último o vetor unitário da aceleração através da velocidade e multiplica-se pela aceleração máxima para dar a do momento. Através da velocidade anterior e da aceleração obtém-se a velocidade no momento e com este calcula-se o vetor unitário da direção. Para que esta rotação seja suave e não brusca calcula-se a rotação no instante, de seguida move-se o tank.

CLASSE: “TankBullet”

Para criar a nossa bullet fizemos o seguinte: Primeiramente criamos um modelo básico de uma esfera com um bone para que nos permita a mover. Além disso, vamos receber como parâmetros, do “BulletManager”, coisas como a sua velocidade, aceleração, direção, posição e tempo decorrido.

```
Model sphere;

Vector3 direction, position, vel, a;
public float t, t0;

Matrix[] boneTransform;

public TankBullet(GraphicsDevice gd, Game1 gm, Vector3 pos, Vector3 dir, Vector3 v, float ti)
{
    this.t0 = ti;
    this.vel = v;
    this.position = pos;
    this.direction = dir * 0.5f;

    vel = direction;

    sphere = gm.Content.Load<Model>("bulletTank");

    boneTransform = new Matrix[sphere.Bones.Count];
```

No update dela fazemos os cálculos que permitam ela se mover da mesma maneira que a noção física “Queda horizontal”, ou seja, ela vai subindo e depois chega a um ponto máximo e finalmente começa a descer.

```
public bool Update(GameTime gt)
{
    t += (float)gt.ElapsedGameTime.TotalSeconds;

    a = Vector3.Down * 9.8f;
    vel += (direction + a) * (float)gt.ElapsedGameTime.TotalSeconds;
    position += vel * (float)gt.ElapsedGameTime.TotalSeconds;

    Matrix scale = Matrix.CreateScale(0.0010f);
    Matrix trans = Matrix.CreateTranslation(position);

    sphere.Root.Transform = scale * trans;

    sphere.CopyAbsoluteBoneTransformsTo(boneTransform);

    return (position.Y < 0 || (t - t0 > 4));
}
```

A direção que obtemos é nos dada da classe do “ClsTank”, que fomos buscar ao próprio modelo do canhão, além de a normalizamos para obter o vetor unitário e obtermos valores mais pequenos e corretos.

```
cannonPos = boneTransform[cannonBone.Index].Translation;
cannonDir = boneTransform[cannonBone.Index].Forward;
cannonDir.Normalize();
```

Temos ainda a “Matrix scale” e a “Matrix trans” que são encarregues de dar o tamanho e o movimento da bala, respetivamente.

No final, se a posição da bala for abaixo do mapa ou tiver passados 4 segundos do disparo da mesma, ele irá depois removê-la no “BulletManager”.

CLASSE: “BulletManager”

Para dar “management” da bullet, criamos um “BulletManager” e é aqui que vamos adicionar, remover, atualizar e desenhar a bullet.

```
public void Update(GraphicsDevice gd, GameTime gt, ClsTank tank)
{
    if (Keyboard.GetState().IsKeyDown(Keys.Space) && tank.player == ClsTank.Player.P1 && isReloading == false)
    {
        bullet.Add(new TankBullet(gd, gm, tank.cannonPos, tank.cannonDir * -80f, new Vector3(0, 10, 0), (float)gt.ElapsedGameTime.TotalSeconds));
        isReloading = true;
    }
    Reloading(gt);

    foreach (TankBullet b in bullet)
    {
        b.Update(gt);
    }

    foreach (TankBullet b in bullet.ToArray())
    {
        if ((b.Update(gt) || b.isColliding) && tank.player == ClsTank.Player.P1)
        {
            gm.colliders.Remove(b);
            bullet.Remove(b);
            Console.WriteLine("Removeu");
        }
    }
}
```

Se clicarmos no espaço ele vai criar a bala com a direção e posição do canhão, que previamente explicamos como obtemos na classe da bullet.

Ele vai fazer o update de cada bala, e depois removê-la caso ela esteja a colidir ou então cumprir os requisitos que falamos em cima no update da mesma. Além disso ele remove o collider antes de a remover para evitar que a bala morra e faça com que o ele se mantenha lá e esteja sempre a perder vida porque o collider está sempre a “acertar” no tank.

Para a bala ter um reload, criamos este método simples. Basicamente, após disparar o “reloadTimer” começa a descer e quando chega a 0, podemos disparar de novo.

```
public void Reloading(GameTime gt)
{
    if (isReloading == true)
    {
        reloadTimer--;
        //Console.WriteLine(reloadTimer);
    }

    if (reloadTimer == 0)
    {
        isReloading = false;
        reloadTimer = 60;
    }
}
```

CLASSE: “ParticleGenerator”

```
public Vector3 Spawn(Vector3 pos)
{
    //Circulo onde poderá dar spawn cada particula
    //361 para fazer [0,360]
    pos = Centro + raio * (float)rnd.NextDouble() * 1.6f * new Vector3((float)Math.Cos(rnd.Next(0, 361)), 0, (float)Math.Sin(rnd.Next(0, 361)));
    return pos;

}
public Vector3 Spawn2(Vector3 centro, Vector3 right, Matrix rot, Vector3 normal)
{
    Vector3 pos;
    Vector3 largura = new Vector3(-0.6f, 0, -0.8f);
    rot = Matrix.CreateTranslation(largura) * rot;
    largura = Vector3.Transform(largura, rot);

    // centro.Y -= normal.Y;
    pos = ((float)rnd.NextDouble() * 2) * -1 * largura;
    pos += centro;
    return pos;
}
```

Para gerar a chuva e a poeira, criamos este “ParticleGenerator” onde basicamente criamos dois spawns diferentes, um para a chuva (o “Spawn”) e outro para a poeira (“Spawn2”). Na chuva apenas achamos um sítio random dentro de um raio circular. Para a poeira achamos um sítio random na largura das rodas e além disso temos uma rotação para que quando o tank virar as partículas continuem a sair atrás das rodas e não noutro sítio qualquer.

CLASSE: “ParticleSystem”

Temos um “ParticleSystem” que vai ser encarregue do “management” das partículas.

```
public void Update(GameTime gt, MapGen mp)
{
    sPart = new List<Particle>();
    int count = rnd.Next(10, 20);
    int n = 0;

    //criador de 100 a 200 particulas por frame
    while (n < count)
    {
        lParticles.Add(new Particle(gParticles.Spawn(Vector3.Zero), Vector3.Zero));
        n++;
    }

    foreach (Particle p in lParticles)
    {
        if (p.active)
        {
            //caso a particula esteja "viva" ele faz update e adiciona na lista de particulas vivas
            p.Update(gt, new Vector3(rnd.Next(-10, 10) * (float)rnd.NextDouble(), -9.8f, rnd.Next(-1, 1) * (float)rnd.NextDouble()), mp);
            sPart.Add(p);
        }
    }
    //apos adicionar todas as particulas "vivas" , ele limpa a lista antiga e vai adicionar as novas outra vez
    lParticles = new List<Particle>();
    foreach (Particle p in sPart)
    {
        lParticles.Add(p);
    }

    GetVertex(lParticles.Count,Color.DarkBlue);
}
```

Neste update, à semelhança da bullet, nesta porção de código temos o ciclo de vida de uma gota de chuva, a sua criação, o seu update e depois a sua remoção com base em 2 listas diferentes, que é a maior diferença entre o código de ambos bullet e este.

```
ic void Update2(GameTime gt, MapGen mp, ClsTank tank)

sPart = new List<Particle>();
int count = rnd.Next(1, 10);
int n = 0;

if (tank.moving)
{
    //criador de 1 a 2 particulas por frame
    while (n < count)
    {
        Matrix rot1 = Matrix.CreateFromAxisAngle(tank.rotacao.Right, (float)rnd.NextDouble()*20-10);
        rot1 = Matrix.CreateFromAxisAngle(tank.rotacao.Up, (float)rnd.NextDouble() * 20 - 10);
        //rotaçao com normal
        Vector3 dir1 = tank.wheelsTransform[3].Backward;
        dir1 = Vector3.Transform(dir1, rot1);

        Matrix rot2 = Matrix.CreateFromAxisAngle(tank.rotacao.Right, (float)rnd.NextDouble() * 20 - 10);
        rot2 = Matrix.CreateFromAxisAngle(tank.rotacao.Up, (float)rnd.NextDouble() * 20 - 10);
        Vector3 dir2 = tank.wheelsTransform[2].Backward;
        dir2 = Vector3.Transform(dir2, rot2);

        lParticles.Add(new Particle(gParticles.Spawn2(tank.boneTransform[tank.wheelsBone[3].Index].Translation, tank.wheelsTransform[3].Right, Matrix.CreateRotationY(tank.wheels[0]), tank.rotacao.Up),
        lParticles.Add(new Particle(gParticles.Spawn2(tank.boneTransform[tank.wheelsBone[2].Index].Translation, tank.wheelsTransform[2].Right, Matrix.CreateRotationY(tank.wheels[0]), tank.rotacao.Up),
        n++;
    }
}
```

Agora para a poeira seguimos o mesmo exemplo, se o tank estiver a mover ele vai criar partículas entre o spawn projetado em cima. Depois vai fazer o mesmo que a chuva faz, atualizar e remover.

```
    foreach (Particle p in lParticles)
    {
        if (p.active)
        {
            //caso a particula esteja "viva" ele faz update e adiciona na lista de particulas vivas
            p.Update(gt, new Vector3(rnd.Next(-10, 10) * (float)rnd.NextDouble(), -9.8f, rnd.Next(-1, 1) * (float)rnd.NextDouble()), mp);
            sPart.Add(p);
        }
    }
    //apos adicionar todas as particulas "vivas" , ele limpa a lista antiga e vai adicionar as novas outra vez
    lParticles = new List<Particle>();
    foreach (Particle p in sPart)
    {
        lParticles.Add(p);
    }

    GetVertex(lParticles.Count,Color.White);
}

public void GetVertex(int length,Color color)
{
    //Criador dos vertices das particulas , neste caso usa a posiçao antiga + a atula para criar  o traço em preto
    vertexArray = new VertexPositionColor[length * 2];
    //A variavel n serve para saber a posição no array que cada vertice da particula tem
    int n = 0;
    foreach (Particle p in lParticles)
    {
        vertexArray[n] = new VertexPositionColor(p.pos, color);
        vertexArray[n + 1] = new VertexPositionColor(p.apos, color);
        //como cada particula tem 2 vertices ele após adicionar os dados soma 2 para serem as seguintes
        n = n + 2;
    }
}
```

Temos o “GetVertex” apenas para a parte dos vértices para depois podermos fazer render das mesmas com base na matéria já explicada nos relatórios anteriores.

INTERFACE: “Collider”

```
public interface Collider
{
    // Obter nome do objeto que tem o collider
    string Name();
    // Notificar objeto que houve colisao
    void CollisionWith(Collider other);
    // Validar colisões (existe/não existe)
    bool CollidesWith(Collider other);
    // Retorna o collider do objeto (Circle)
    Collider GetCollider();
}
```

A interface “Collider” serve para que todos que herdarem dela, precisem de ter essas exatas funções.

CLASSE: “SphereCollider”

```
virtual public bool CollidesWith(SphereCollider other)
{
    float dist1 = (Center - other.Center).LengthSquared();
    float dist2 = (float)Math.Pow(Radius + other.Radius, 2f);
    return dist2 >= dist1;
}
```

Neste método vamos achar se algo está a colidir com outra coisa, se a distância de algo for igual ou estiver “dentro” da outra, ele vai dizer que está a colidir.

```
    public string Name() { return "undef"; }

    public void CollisionWith(Collider other) { }
```

Aqui a string vai retornar um nome.

O “CollisionWith” vai servir para nós dizermos se o tank colide com o outro. Além disso, quando eles colidem, ou seja, quando entra na zona do outro círculo, ele volta para a posição anterior.

```
if (other.Name() == "Tank")
{
    position = bPos;
}
```

CLASSE: “Segment”

Para podermos fazer a interseção da bala com o tank, usamos a fórmula de Heron para que não ocorra casos onde a bala pode frame a frame, por ser muito rápida, acabe por dar uma espécie de salto e acabar por nem tocar no tank e então não haver colisão, o que aconteceria caso fosse apenas a deteção de 2 círculos como fazemos no tank.

```
virtual public bool CollidesWith(SphereCollider other)
{
    double a = Distancia(other.Center, pos2);
    double b = Distancia(other.Center, pos1);
    double c = Distancia(pos1, pos2);
    double sp = (a + b + c) / 3;

    double area = Math.Sqrt(sp * (sp - a) * (sp - b) * (sp - c));
    double d = 2 * area / c;

    if (d < other.Radius)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

O método que nós usamos aí, a “Distancia”, serve para calcular a distância entre 2 pontos.

```
private double Distancia(Vector3 p1, Vector3 p2)
{
    float xdif = (p1.X - p2.X) * (p1.X - p2.X);
    float ydif = (p1.Y - p2.Y) * (p1.Y - p2.Y);
    float zdif = (p1.Z - p2.Z) * (p1.Z - p2.Z);
    double dist = Math.Sqrt(xdif + ydif + zdif);

    return dist;
}
```

CLASSE: “Range”

```
class Range
{
    public float radius;
    public Vector3 center;

    public Range(float r, Vector3 pos)
    {
        radius = r;
        center = pos;
    }

    public void Update(Vector3 position)
    {
        center = position;
    }
    //Descobrir se raio colide com raio
    public bool OnRange(Range range)
    {
        float xdif = (center.X - range.center.X) * (center.X - range.center.X);
        float ydif = (center.Y - range.center.Y) * (center.Y - range.center.Y);
        float zdif = (center.Z - range.center.Z) * (center.Z - range.center.Z);
        double difference = Math.Sqrt(xdif + ydif + zdif);

        return difference < radius;
    }
    //Descobrir se esta dentro do raio usando a posição
    public bool OnRangePos(Vector3 pos)
    {
        float xdif = (center.X - pos.X) * (center.X - pos.X);
        float ydif = (center.Y - pos.Y) * (center.Y - pos.Y);
        float zdif = (center.Z - pos.Z) * (center.Z - pos.Z);
        double difference = Math.Sqrt(xdif + ydif + zdif);

        return difference < radius;
    }
}
```

Esta classe vai servir para o modo CPU, mais precisamente para saber quando o tank do CPU pode perseguir o do player, ou seja, calculamos uma espécie de collider para verificar se um está dentro do raio do outro. Depois no tank temos o código da perseguição em si, o movimento.

Conclusão

Com este trabalho começamos a aprender a programar em 3D, usando trigonometria para tal feito, o que por si, é um grande desafio porque não é algo relativamente fácil.

Apesar disso acho que consolidamos alguns conceitos e aprendemos um pouco melhor a transformação que é passar do 2D para o 3D.

Houve dificuldades, como já referimos, em alguns cálculos, mas no geral acreditamos que conseguimos cumprir o necessário.