Curso: Desenvolvendo Jogos 2d Com C# E Microsoft XNA

Conteudista: André Luiz Brazil

Aula 9: TRATANDO COLISÕES DE OBJETOS NO JOGO

META

Fazer desaparecer da tela do jogo a espaçonave inimiga quando um tiro acertá-la.

OBJETIVOS

Ao final da aula, você deve ser capaz de:

1. Verificar se um tiro acertou uma espaçonave do jogo.

PRÉ-REQUISITOS

1. Conhecer a ferramenta XNA Game Studio, conceito abordado na aula 2;
2. Possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na aula 3**;**
3. Ter o seu projeto de jogo atualizado conforme o conteúdo da aula 7, que inclui a produção de tiros pela espaçonave.

Introdução

Na aula anterior, incorporamos diversos sons aos acontecimentos do nosso jogo.

Figuras 9.1 e 9.2 – Exemplos de colisão (Jefferson, favor redesenhar)

Fontes:

- http://cheezeworld.com/wp-content/uploads/2008/11/collision.jpg
- http://ps2media.gamespy.com/ps2/image/article/690/690758/ea-sports-fight-night-round-3-20060222065334991-000.jpg

Lembra-se da lei da física que diz que dois objetos não podem ocupar o mesmo lugar no espaço ao mesmo tempo?

Exatamente! Para trazermos um pouco mais de realismo ao jogo, agora vamos aprender como tratar as colisões entre os objetos do jogo.

Verificando colisões simples entre os objetos do jogo

Vejamos agora como funciona um teste de colisão simples.

Caixa de Ênfase

Uma **colisão simples** ocorre quando a área ocupada por uma textura é invadida por alguma outra textura do jogo.

Fim da Caixa de Ênfase

A **Figura 9.3** nos mostra um exemplo de **colisão simples** entre duas texturas dentro do jogo:

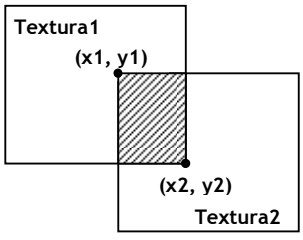

Figura 9.3 – Colisão simples entre duas texturas

A verificação de uma **colisão simples** no jogo é bem fácil. Repare que as texturas das naves e dos tiros são todas retangulares. Precisamos apenas testar se o retângulo correspondente à primeira textura intercepta o retângulo referente a uma segunda textura.

Vamos criar um método chamado **ColisaoSimples** para realizar esta verificação. O método precisará receber como parâmetros as duas texturas a serem verificadas e também as posições onde as texturas estão localizadas na tela do jogo.

```
public bool ColisaoSimples(Texture2D Textura1, Texture2D Textura2, int
                 PosicaoX1, int PosicaoY1, int PosicaoX2, int PosicaoY2)
{
     return (PosicaoX1 + Textura1.Width > PosicaoX2 &&
             PosicaoX1 < PosicaoX2 + Textura2.Width &&
             PosicaoY1 + Textura1.Height > PosicaoY2 &&
             PosicaoY1 < PosicaoY2 + Textura2.Height);
}
```

Observe que o método **ColisaoSimples** retorna um resultado lógico (verdadeiro ou falso), indicando se houve ou não a colisão entre as texturas. O método realiza quatro verificações e se todas forem verdadeiras, a colisão terá ocorrido. As verificações são:

- A posição x da textura 1 mais a largura da textura 1 ultrapassam a posição x da textura2?
- A posição x da textura 2 mais a largura da textura 2 ultrapassam a posição x da textura1?
- A posição y da textura 1 mais a altura da textura 1 ultrapassam a posição y da textura2?
- A posição y da textura 2 mais a altura da textura 2 ultrapassam a posição y da textura1?

Para testarmos as colisões, precisamos criar também uma espaçonave inimiga dentro do nosso jogo e fazer alguns ajustes:

Passo 1: Acrescente o atributo **NaveInimiga** aos atributos do jogo:

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
```

```
// Definindo a espaçonave do jogador no jogo
Espaconave NaveJogador;

// Definindo a espaçonave inimiga no jogo
Espaconave NaveInimiga;
```

Passo 2: Precisamos acrescentar à classe Espaconave alguns métodos para configurar as posições x e y da espaçonaves e retornar a textura:

```
public int PosicaoX()
{
    return _posicaox;
}

public int PosicaoY()
{
    return _posicaoy;
}

public void PosicaoX(int p_posicaox)
{
    _posicaox = p_posicaox;
}

public void PosicaoY(int p_posicaoy)
{
    _posicaoy = p_posicaoy;
}

public Texture2D Textura()
{
    return _textura;
}
```

Passo 3: Assim como a espaçonave do jogador, a nave inimiga também possui um nome e uma posição inicial. Configure as espaçonaves do jogador e inimiga no método de inicialização do jogo (**Initialize**):

```
protected override void Initialize()
{
      // Criando efetivamente a espaçonave do jogador
      NaveJogador = new Espaconave();

      // Configurado a nave do jogador
      NaveJogador.Nome("Falcão Justiceiro");
      NaveJogador.PosicaoX(300);
      NaveJogador.PosicaoY(300);

      // Criando efetivamente a espaçonave inimiga
      NaveInimiga = new Espaconave();

      // Configurando a nave inimiga
      NaveInimiga.Nome("Soldado do Espaço");
      NaveInimiga.PosicaoX(200);
      NaveInimiga.PosicaoY(10);
```

Passo 4: A nave inimiga também precisa aparecer na tela do jogo. Carregue uma textura para ela dentro do método de carga de conteúdo (**LoadContent**) do jogo e desenhe ela no método de exibição do jogo (**Draw**):

```
protected override void LoadContent()
{
     // Create a new SpriteBatch, which can be used to draw textures.
     spriteBatch = new SpriteBatch(GraphicsDevice);

     // TODO: use this.Content to load your game content here
     NaveJogador.CarregarTextura(graphics.GraphicsDevice,
      "../../../Content/Imagens/nave.PNG");
     NaveInimiga.CarregarTextura(graphics.GraphicsDevice,
      "../../../Content/Imagens/inimigo.PNG");


protected override void Draw(GameTime gameTime)
{
            GraphicsDevice.Clear(Color.Black);

            // Desenhando o cenário de fundo do jogo
            spriteBatch.Begin();
            spriteBatch.Draw(cenario_fundo, new Rectangle(0,
            posicaoy_cenario_fundo, this.Window.ClientBounds.Width,
            this.Window.ClientBounds.Height), Color.White);
            spriteBatch.Draw(cenario_fundo, new Rectangle(0,
            posicaoy_cenario_fundo - this.Window.ClientBounds.Height,
            this.Window.ClientBounds.Width, this.Window.ClientBounds.Height),
            Color.White);
            spriteBatch.End();

            // TODO: Add your drawing code here
            NaveJogador.Desenhar(spriteBatch);
            NaveInimiga.Desenhar(spriteBatch);
```

Lembre-se de adicionar a imagem da nave inimiga à pasta imagens, localizada dentro da pasta **Content** do seu projeto do jogo. Clique com o botão direito sobre a pasta imagens dentro do **Solution Explorer** e utilize a opção **Add**/**Existing Item** para acrescentá-la ao seu projeto de jogo.

Passo 5: Vamos agora testar a colisão entre os tiros e a espaçonave inimiga. Ajuste o método de atualização do jogo (**Update**) da seguinte forma após a movimentação dos tiros:

```
      // Movendo os tiros do jogo
      foreach (Tiro oTiro in TirosDisparados.GetRange(0,TirosDisparados.Count))
      {
                oTiro.Mover();

               // Testando a colisão do tiro com o inimigo
               if (ColisaoSimples(oTiro.Textura(),
                   NaveInimiga.Textura(),
                   oTiro.PosicaoX(), oTiro.PosicaoY(),
```

```
            NaveInimiga.PosicaoX(),
            NaveInimiga.PosicaoY()))
        {
            // Removendo o tiro
            TirosDisparados.Remove(oTiro);

            // Removendo a espaçonave inimiga da tela
            NaveInimiga.PosicaoX(-100);
            NaveInimiga.PosicaoY(-100);
        }
```

Repare que é utilizado um artifício para fazer a espaçonave inimiga desaparecer da tela do jogo. As posições x e y da nave inimiga são atualizadas para o valor -100, localizado fora da tela do jogo.

Excelente! Agora você já é capaz de verificar todas as colisões de objetos dentro do seu projeto de jogo. Execute o jogo e veja o resultado.

Atividade Prática 1 – Atende ao Objetivo 1

Procure uma imagem para a espaçonave inimiga. Seguindo os procedimentos descritos anteriormente, abra o seu projeto de jogo na ferramenta Visual C# e acrescente ao seu jogo a espaçonave inimiga e uma verificação de **colisões simples** entre os tiros disparados e a espaçonave inimiga, fazendo a nave inimiga e o tiro desaparecerem da tela do jogo após a colisão.

Fim da Atividade Prática 1

Verificando colisões avançadas entre os objetos do jogo

Vamos agora aprender a verificar colisões de uma forma mais detalhada.

O método **ColisaoSimples** resolve a maioria das colisões existentes. Contudo, algumas texturas possuem uma parte transparente. Para estes casos, precisaremos utilizar um teste mais avançado de colisão das texturas.

Para o teste avançado de colisão de texturas, estaremos verificando:

- Se as áreas das texturas estão sobrepostas – **ColisaoSimples**;
- Caso estejam, se dentro da área sobreposta, existe algum ponto não transparente da textura1 sobrepondo outro ponto não transparente da textura2.

Veja como implementamos o teste avançado de colisões:

```
public bool ColisaoAvancada(Texture2D Textura1,Texture2D Textura2, int
                            PosicaoX1, int PosicaoY1, int PosicaoX2, int
                            PosicaoY2)
{
            if (ColisaoSimples(Textura1, Textura2, PosicaoX1, PosicaoY1,
                               PosicaoX2, PosicaoY2))
            {
                // Obtendo os bits das texturas
                uint[] BitsTextura1, BitsTextura2;
                BitsTextura1 = new uint[Textura1.Width * Textura1.Height];
                BitsTextura2 = new uint[Textura2.Width * Textura2.Height];

                Textura1.GetData<uint>(BitsTextura1);
                Textura2.GetData<uint>(BitsTextura2);

                // Obtendo as coordenadas x e y mínima e máxima
                // de intersecção entre as texturas
                int x1 = Math.Max(PosicaoX1, PosicaoX2);
                int y1 = Math.Max(PosicaoY1, PosicaoY2);

                int x2 = Math.Min(PosicaoX1 + Textura1.Width,PosicaoX2 +
                                  Textura2.Width);
                int y2 = Math.Min(PosicaoY1 + Textura1.Height,PosicaoY2 +
                                  Textura2.Height);

                // Percorrendo a àrea de intersecção das Naves
                // para verificar se os pixels são transparentes ou não
                for (int linha = y1; linha < y2; linha++)
                {
                    for (int coluna = x1; coluna < x2; coluna++)
                    {
                        // Se ambos os pixels das naves A e B verificados
                        // tem cor então = Colisão
                        if (    ((BitsTextura1[(coluna - PosicaoX1) + (linha -
                                  PosicaoY1) * Textura1.Width]
                                  & 0xFF000000) >> 24) > 20 &&
                                ((BitsTextura2[(coluna - PosicaoX2) + (linha -
                                  PosicaoY2) * Textura2.Width]
                                  & 0xFF000000) >> 24) > 20)
                        {
                            return true;
                        }
                    }
                }
            }
            return false;
}
```

Repare que a primeira verificação realizada dentro do método **ColisaoAvancada** é chamar o método **ColisaoSimples**, para verificar há alguma sobreposição das texturas.

Caso haja essa sobreposição, é utilizado o método **GetData** da classe de texturas, para extrair as cores dos pontos existentes nas duas texturas. Estas cores são armazenadas dentro de duas listas, que são **BitsTextura1** e **BitsTextura2**.

Por fim, toda a área sobreposta entre as duas texturas é percorrida utilizando dois comandos de repetição (**for**) encadeados. Dentro deles é realizado um teste para ver se

em alguma parte da área sobreposta, ambos os pontos são coloridos, ou seja, se o valor da cor de uma posição na lista **BitsTextura1** é maior que 20 e o valor da cor nesta mesma posição na lista **BitsTextura2** também é maior que 20. Isto configura a colisão avançada de texturas.

Vamos agora modificar o nosso teste de colisão entre os tiros disparados e a espaçonave inimiga no método de atualização do jogo (**Update**), trocando a chamada **ColisaoSimples** por **ColisaoAvancada**:

```
// Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados.GetRange(0,TirosDisparados.Count))
      {
          oTiro.Mover();

          // Testando a colisão do tiro com a nave inimiga
          if (ColisaoAvancada(oTiro.Textura(),
                              NaveInimiga.Textura(), oTiro.PosicaoX(),
                              oTiro.PosicaoY(), NaveInimiga.PosicaoX(),
                              NaveInimiga.PosicaoY()))
          {
              // Removendo o tiro
              TirosDisparados.Remove(oTiro);
              // Removendo a espaçonave inimiga da tela do jogo
              NaveInimiga.PosicaoX(-100);
              NaveInimiga.PosicaoY(-100);
          }
```

Atividade Prática 2 – Atende ao Objetivo 1

Seguindo os procedimentos descritos anteriormente, abra o seu projeto de jogo na ferramenta Visual C# e acrescente ao seu jogo a verificação de **colisões avançadas** entre os tiros e a espaçonave inimiga, fazendo a nave inimiga e o tiro desaparecerem da tela do jogo após a colisão.

Fim da Atividade Prática 2

Atividade Prática 3 – Atende ao Objetivo 1

Agora que você já sabe verificar as colisões, programe dentro do seu projeto de jogo um teste de **colisão avançada** entre a espaçonave do jogador e a espaçonave inimiga.

Fim da Atividade Prática 3

CAIXA DE FÓRUM **Informação sobre Fórum**

Figura 9.4
Fonte: http://www.stockxpert.com/browse_image/view/28331341/?ref=sxc_hu (Jefferson- favor redesenhar)

Você teve alguma dificuldade para verificar as colisões dentro do jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os seus amigos.

FIM DE CAIXA DE FÓRUM

CAIXA DE ATIVIDADE **Informação sobre Atividade on-line**

Figura 9.5
Fonte: http://www.sxc.hu/photo/1000794 (Jefferson : favor redesenhar)

Agora que você já está com o seu código de projeto do jogo ajustado para verificar colisões entre objetos do jogo, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

FIM CAIXA DE ATIVIDADE

Resumo

- Uma **colisão simples** ocorre quando a área ocupada por uma textura é invadida por alguma outra textura do jogo.
- Para verificar colisões entre objetos no seu projeto de jogo, você precisa testar se o retângulo correspondente à primeira textura intercepta o retângulo referente a uma segunda textura.
- É necessário acrescentar uma espaçonave inimiga ao jogo para realizar estas verificações. Isto implica em:
    - Acrescentar o atributo **NaveInimiga** ao código de criação do jogo;
    - Configurar o atributo **NaveInimiga** dentro do método de inicialização do jogo (**Initialize**);
    - Acrescentar os métodos **PosicaoX**, **PosicaoY** e **Textura** à classe Espaconave para obter e configurar a posição das espaçonaves e obter a textura da espaçonave;

    - Carregar a textura da espaçonave inimiga no método de carga de conteúdo do jogo (**LoadContent**);
    - Desenhar a espaçonave inimiga no método de exibição do jogo (**Draw**).
- É necessário também acrescentar o teste de colisão simples entre o tiro disparado e a espaçonave inimiga no método de atualização do jogo (**Update**).
- Para o teste avançado de colisão de texturas, estaremos verificando:
    - Se as áreas das texturas estão sobrepostas – **ColisaoSimples**;
    - Caso estejam, se dentro da área sobreposta, existe algum ponto não transparente da textura1 sobrepondo outro ponto não transparente da textura2.

Fim do resumo

Informações sobre a próxima aula

Na próxima aula, veremos como produzir efeitos duradouros, tais como explosões, dentro do jogo.