TI24X7

CURSOS ONLINE EM TECNOLOGIA

Curso XNA Desenvolvimento de jogos

Módulo 5

Autor: Fernando Amaral

WWW.TI24X7.COM.BR

5. Detectando colisão

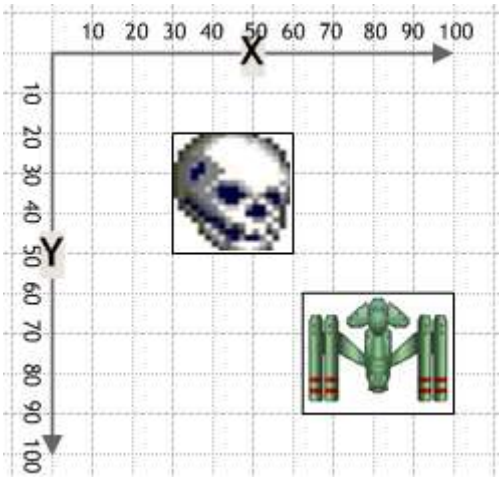
Você provavelmente não conseguirá fazer um jogo, por mais simples que seja, se não for implementado algum tipo de mecanismo de detecção de colisão. Como saber se o tiro atingiu o inimigo? Este é um dos algoritmos mais importantes na confecção de qualquer jogo. Neste capitulo vamos ver como a detecção de colisão funciona, e vamos implementar alguns mecanismos básicos de detecção e até criar um jogo muito simples.

Tipos de detecção de Colisão

Existem várias formas de se detectar colisão entre sprites do seu jogo. As mais comuns são a detecção retangular e a detecção em nível de pixel. Na detecção retangular, o seu código deve verificar se a sobreposição de uma imagem sobre a outra, considerando toda a extensão da imagem. Na detecção em nível de pixel, a detecção tem maior precisão, pois serão verificados apenas os pixels que compõe a imagem. Existem ainda outras técnicas, como dividir o sprite em vários retângulos e verificar a colisão em cada um deles. Quanto mais precisa a técnica, mas processamento será necessário, porém, mais preciso será o resultado.

Detectando colisões entre sprites retangulares

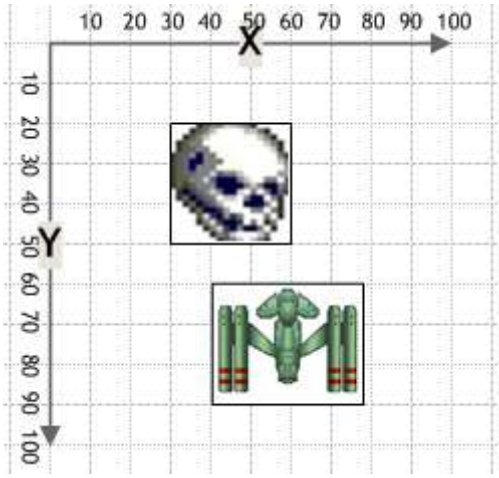
Em nosso curso vamos estudar a detecção de colisão retangular. Para facilitar o entendimento, vamos imaginar a tela de um jogo, com suas coordenadas no eixo X e dois sprites, uma caveira medindo 30 x 30, e uma nave espacial medindo 40 X 30. Para facilitar mais o entendimento, as imagens são desenhadas com suas bordas, pois estas são as medidas que serão consideradas em uma detecção de colisão do tipo retangular.



Para detectar a colisão, temos que verificar quatro condições: duas relativas ao eixo X e duas relativas o eixo Y. Se todas as condições forem verdadeiras, significa que esta ocorrendo uma colisão entre as imagens. As quatro condições são:

1. **A posição da caveira no eixo X + sua largura deve ser maior que a posição da nave no eixo X;**
2. **A posição da caveira no eixo X deve ser menor que a posição da nave no eixo X + a largura da nave;**
3. **A posição da caveira no eixo Y mais a altura da caveira devem ser maior que a posição da nave no eixo Y;**
4. **A posição da caveira no eixo Y deve menor que a posição da nave no eixo Y + altura da nave.**

Confuso? Vamos executar alguns “testes de mesa” e algoritmo a ser construído ficará claro. Primeiro, imagine uma tela onde não há colisão entre os sprites:



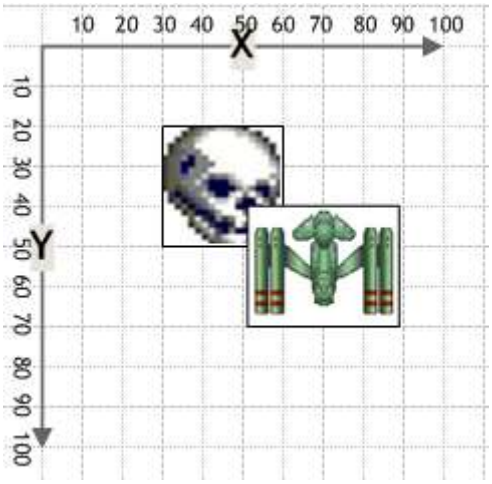
Agora vamos verificar as quatro condições fazendo os respectivos cálculos:

Condição	Cálculo
1. A posição da caveira no eixo X + sua largura deve ser maior que a posição da nave no eixo X;	30 + 30 > 40 Verdadeiro!
2. A posição da caveira no eixo X deve ser	30 < 40 + 40 Verdadeiro!

menor que a posição da nave no eixo X + a largura da nave;	
3. A posição da caveira no eixo Y mais a altura da caveira deve ser maior que a posição da nave no eixo Y;	20 + 30 > 60 Falso!
4. A posição da caveira no eixo Y deve menor que a posição da nave no eixo Y + altura da nave.	20 < 60 + 30 Verdadeiro!

Vejam que no nosso teste de mesa, a condição 3 foi falsa. Lembre-se: temos que ter as quatro condições verdadeiras para haver colisão!

Vamos ao um segundo teste onde o resultado deve definir colisão:



Condição	Cálculo
1. A posição da caveira no eixo X + sua largura deve ser maior que a	30 + 30 > 50 Verdadeiro!

posição da nave no eixo X;	
2. A posição da caveira no eixo X deve ser menor que a posição da nave no eixo X + a largura da nave;	30 < 50 + 40 Verdadeiro!
3. A posição da caveira no eixo Y mais a altura da caveira deve ser maior que a posição da nave no eixo Y;	20 + 30 > 40 Verdadeiro!
4. A posição da caveira no eixo Y deve menor que a posição da nave no eixo Y + altura da nave.	20 < 40 + 30 Verdadeiro!

O desenho nos mostra que há colisão. O teste de mesa confirma o óbvio, com as quatro condições verdadeiras, temos uma colisão!

Agora com implementar isto no XNA? É o que vamos fazer na próxima seção.

Detectando colisão na prática

Para colocar em prática vamos construir um pequeno jogo que vai juntar todo o que vimos até agora. A idéia é ter dois sprites: Uma caveira que se moverá pela tela, colidindo com as bordas, e uma nave espacial, controlada pelo usuário. Nosso jogo terá um sistema de pontos que será incrementado automaticamente em nosso método Update. Porém, havendo colisão, os pontos serão reduzidos de forma muito mais veloz. Para darmos um pouco mais de emoção ao jogo, vamos inserir um pequeno código que fará com que a caveira mude de direção aleatoriamente, mesmo sem se chocar com uma borda da tela.

Abra o Visual Studio e crie um novo projeto de Jogo Windows Game. De o nome de Colisao.

No projeto de recursos adicione uma pasta Imagens e outra Fontes. Vamos precisar da fonte para escrever a pontuação na tela. Na pasta fontes criada, clique com o botão direito, selecione Add, New Item, Sprite Font e mantenha o nome padrão. Na pasta Imagens, selecione Add, Existem Item. Localize as imagens Caveira.bmp e NaveEspacial no diretório de exemplo do livro.

Vamos à codificação. Adicione em nível de classe uma variável caveira, do tipo Texture2D, uma variável vetorcaveira, do tipo Vector2 e uma variável velocidadecaveira, do tipo Vector2. Estas variáveis servirão para o sprite, a posição e a velocidade da caveira, respectivamente. A variável velocidade já é inicializada na declaração com valores 6,6. Você pode alterar estes valores para aumentar ou diminuir a dificuldade do jogo. Adicione agora uma variável nave, do tipo Texture2D, e uma variável vetornave, do tipo Vector2. Estas variáveis servirão para o sprite e a posição da nave, respectivamente. Além disto, vamos precisar de uma variável fonte, do tipo SpriteFont, para escrever a pontuação na tela, uma variável do tipo inteiro, pontos, para guardar a pontuação do jogo, e finalmente uma variável do tipo inteiro, numaleatorio, para o numero aleatório que poderá mudar a direção da caveira e dar mais emoção ao jogo.

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

    Texture2D caveira;
    Vector2 vetorcaveira;
    Vector2 velocidadecaveira = new Vector2(6, 6);

    Texture2D nave;
    Vector2 vetornave;


    SpriteFont fonte;
    int pontos;
    int numaleatorio;
```

Vamos agora implementar duas funções. A primeiro, chamada GeraAleatorio, recebe com parâmetro um valor mínimo e Maximo e retorna um número aleatório dentro deste intervalo (na verdade menor que o valor de Max). A função faz um uso simples da função Random do .NET e será utilizada mais adiante, no nosso método Update.

```
private int GeraAleatorio(int min, int max)
 {
     Random random = new Random();
     return random.Next(min, max);
 }
```

A próxima função, nada mais é do que implementar nosso algoritmo de detecção de colisão. Se você analisar, verá que simplesmente traduzimos em C# as quatro funções obrigatórias

para termos uma colisão. Como as quatro são obrigatórias, usamos o operador lógico && para obtermos um resultado "verdade" se ambas as quatro condições forem verdadeiras:

```
public bool DetectaColisao()
{
   return (vetorcaveira.X + caveira.Width > vetornave.X &&
    vetorcaveira.X < vetornave.X + nave.Width &&
    vetorcaveira.Y + caveira.Height > vetornave.Y &&
    vetorcaveira.Y < vetornave.Y + nave.Width);
}
```

Agora nosso método LoadContent. Aqui nada de novo, carregamos os dois sprites, a nossa fonte, inicializamos a posição da caveira e da nave. Para tentar deixar o jogo justo, posicionamos a caveira na extremidade superior esquerda da tela, enquanto nossa nave espacial deverá ocupar exatamente o centro da tela:

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

       caveira = Content.Load<Texture2D>(@"Imagens\Caveira");
       nave = Content.Load<Texture2D>(@"Imagens\NaveEspacial");
       fonte = Content.Load<SpriteFont>(@"Fontes\SpriteFont1");

       vetorcaveira = new Vector2(10, 10);
       vetornave = new Vector2(Window.ClientBounds.Width / 2 - nave.Width /
              2, Window.ClientBounds.Height / 2 - nave.Height / 2);


       // TODO: use this.Content to load your game content here
  }
```

Agora examinamos o método Draw, aqui também nenhuma novidade: Desenhamos a nave, a caveira e o texto, nas posições definidas nas variáveis:

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

   spriteBatch.Begin();
   spriteBatch.Draw(nave, vetornave,Color.White);
   spriteBatch.Draw(caveira, vetorcaveira, Color.White);
   spriteBatch.DrawString(fonte,  pontos.ToString(), Vector2.Zero,
             Color.White);
   spriteBatch.End();

    base.Draw(gameTime);
}
```

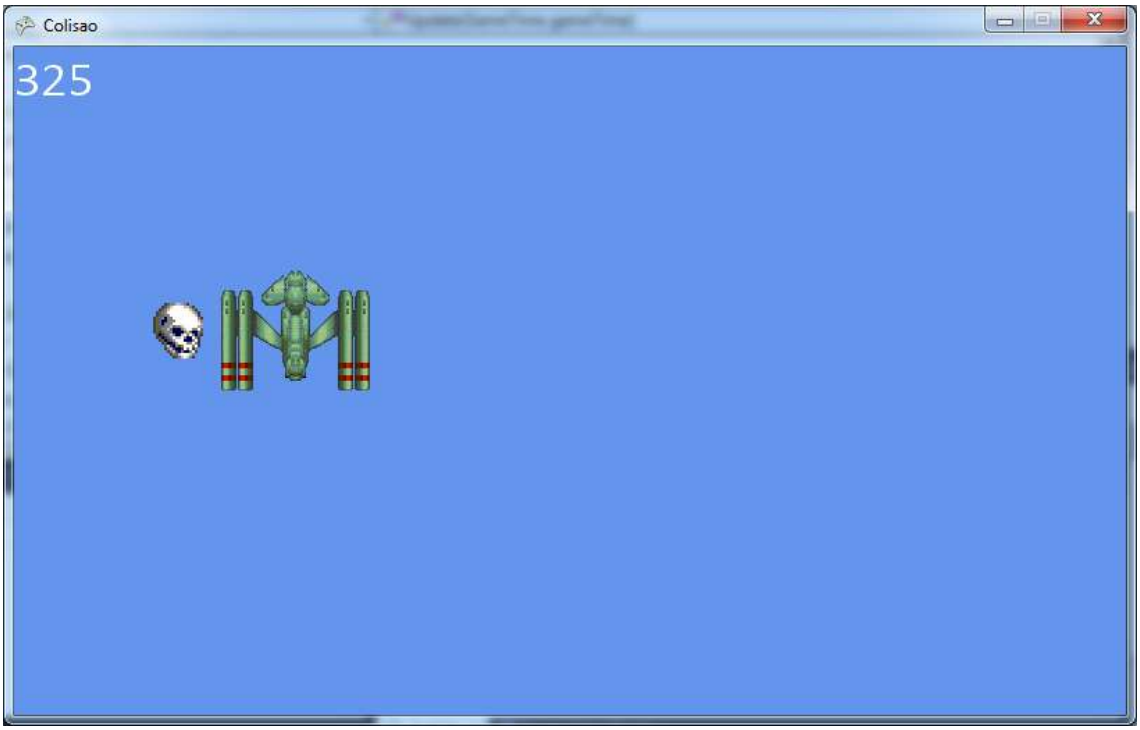
Vamos agora ao nosso método Update, onde a emoção acontece! O primeiro bloco de código detecta as teclas e movimento a Nave Espacial. O segundo bloco movimenta a caveira dentro dos limites da tela, fazendo-a quicar nas bordas. Em seguida, nossa função GeraAleatorio é chamada, um bloco switch define que caso o valor seja 2, a caveira vai mudar sua direção vertical, caso seja 3, muda sua posição horizontal, e caso seja 4, a mudança de direção será nos dois sentidos. Esta foi uma definição totalmente aleatória, você pode ajustar com quiser, porém lembre-se que o método Update é atualizado dezenas de vezes enquanto o seu sprite se movimenta, portanto, não tente usar, por exemplo, um valor Maximo de 10 para chamar a função. Finalmente, a função de detecção de colisão é chamada. Caso ela ocorra, 15 pontos são perdidos, caso não ocorra, um ponto é adicionado. Aqui vale a mesma observação: O método Update é executado dezenas de vezes enquanto nossos sprites são desenhados, portanto, alguns segundos em colisão poderão fazer seus pontos ficarem rapidamente negativos.

```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
                    ButtonState.Pressed)
        this.Exit();

    KeyboardState teclado = Keyboard.GetState();
    //Verifica se alguma tecla de movimento esta pressionada
    if (teclado.IsKeyDown(Keys.Left))
           vetornave.X -= 3;
    if (teclado.IsKeyDown(Keys.Right))
            vetornave.X += 3;
    if (teclado.IsKeyDown(Keys.Up))
           vetornave.Y -= 3;
    if (teclado.IsKeyDown(Keys.Down))
            vetornave.Y += 3;

    //detecta limites horizontais da tela e inverte a velocidade
    if (vetorcaveira.X > Window.ClientBounds.Width - caveira.Width ||
                    vetorcaveira.X < 0)
        velocidadecaveira.X *= -1;
    //detecta limites verticais da tela e inverte a velciodade
    if (vetorcaveira.Y > Window.ClientBounds.Height - caveira.Height ||
                    vetorcaveira.Y < 0)
        velocidadecaveira.Y *= -1;

   /Muda a direção da caveira, aleatóriamente
   numaleatorio = GeraAleatorio(1, 100);
   switch (numaleatorio)
   {   case 2:
           velocidadecaveira.Y *= -1;
           break;
       case 3:
           velocidadecaveira.X *= -1;
           break;
       case 4:
           velocidadecaveira.Y *= -1;
           velocidadecaveira.X *= -1;
           break;
```

```
    }


     //atualiza a posição do sprite
     vetorcaveira.X += velocidadecaveira.X;
     vetorcaveira.Y += velocidadecaveira.Y;

     //detecta colisão, reduz pontos em caso de verdadeiro,
     //incrementa em caso negativo
     if (DetectaColisao())
     pontos -= 15;
     else
     pontos +=1;

     base.Update(gameTime);
}
```

Finalmente, você deve liberar os recursos carregados no método LoadContent.

Rode a aplicação. Note que a caveira se movimenta de forma mais veloz e aleatoriamente muda a direção. Movimente a nave tentando evadir-se. Observe a pontuação: mesmo parecendo incrementar rapidamente, qualquer fração de tempo havendo colisão poderá deixá-lo com sua preciosa pontuação negativa.



Parabéns, você construiu seu primeiro jogo em XNA! Esta longe ainda de um sucesso de publico, no máximo seu irmão irá jogar alguns minutos em consideração ao seu esforço, mas pense o quanto já evoluímos desde o inicio dos estudos!

Conclusão

Com a detecção de colisão entendida, tudo o que falta para termos todos os elementos para a criação de jogos de verdade é...Som! Este é o assunto do próximo capitulo.