TI24X7

CURSOS ONLINE EM TECNOLOGIA

Curso XNA Desenvolvimento de jogos

Módulo 4

Autor: Fernando Amaral

WWW.TI24X7.COM.BR

4. Dando o controle ao jogador

Do ponto de vista de criação, o que diferencia uma simples animação de um jogo? Muita coisa, claro, mas principalmente a interação do usuário e a detecção de colisão. Detectar colisão é assunto para o próximo capitulo. Neste, vamos ver como interagir com seu jogo controlando um sprite através do teclado ou do mouse. Vamos começar pelo teclado.

Para nosso exemplo, crie um novo projeto XNA do tipo Windows Game, de o nome de Controle. Primeiro vamos adicionar uma imagem para termos um sprite para controlar. Adicione uma pasta Imagens no projeto de recursos, adicione a imagem NaveEspacial usada no capitulo 3 – ou qualquer outra imagem de sua preferência.

Agora vamos ao projeto. Declare em nível de classe um tipo Texture2D para o sprite e um tipo Vector2 para a posição do mesmo. Declare ainda uma constante do tipo inteiro para a velocidade com o sprite vai se mover. Vamos atribuir inicialmente um valor igual a 3.

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

    Texture2D sprite;
    Vector2 vetor;
    const int velocidade = 3;
```

No método LoadContent, carregue seu sprite. Em seguida vamos definir a posição inicial do sprite. Para centralizá-lo, buscamos a metade da largura da tela menos a metade da largura do sprite. Para a altura usamos o mesmo principio:

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

    sprite = Content.Load<Texture2D>(@"Imagens\NaveEspacial");
    vetor = new Vector2(Window.ClientBounds.Width / 2 - sprite.Width / 2
            , Window.ClientBounds.Height /2 - sprite.Height / 2);

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

No método Draw desenhe o nosso sprite, usando a variável vetor para posicioná-lo no centro da tela:

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

    // TODO: Add your drawing code here
```

```
        spriteBatch.Begin();
        spriteBatch.Draw(sprite, vetor, Color.White);
        spriteBatch.End();

        base.Draw(gameTime);
    }
```

Até aqui não tivemos nenhuma novidade! Se você rodar a aplicação agora verá nossa nova espacial imutável, no centro da tela.

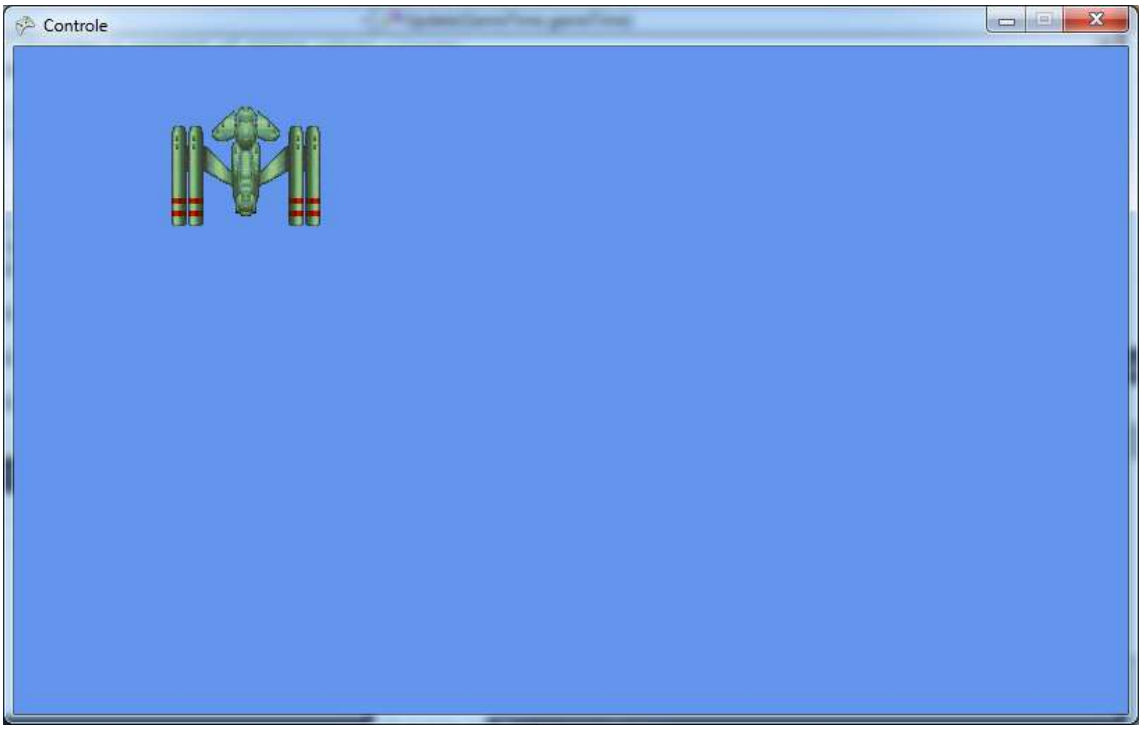
Vamos agora as novidades. Para movermos o sprite, devemos simplesmente alterar os valores da variável vetor, de acordo com as teclas pressionadas. O mais usual em jogos é usarmos as telas de setas para esta função. Primeiramente declaramos uma variável do tipo Keybordstate e atribuímos a ela o estado atual do teclado, que é capturado no momento da execução do método Update. Em seguida, basta verificarmos se na captura realizada uma das teclas que nos interessam estavam pressionadas, alterando o valor do vetor. Como resultado, no método Draw, o sprite será desenhado com a nova posição atribuída na variável vetor:

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

        //captura o estado do teclado
        KeyboardState teclado = Keyboard.GetState();
        //Verifica se alguma tecla de movimento esta pressionada
        if (teclado.IsKeyDown(Keys.Left))
            vetor.X -= velocidade;
        if (teclado.IsKeyDown(Keys.Right))
            vetor.X += velocidade;
        if (teclado.IsKeyDown(Keys.Up))
            vetor.Y -= velocidade;
        if (teclado.IsKeyDown(Keys.Down))
            vetor.Y += velocidade;


        base.Update(gameTime);
}
```

Rode a aplicação e teste o movimento do sprite com as teclas de setas. Se você quiser aumentar a velocidade do movimento, basta alterar o valor da constante velocidade na sua declaração. Observe ainda que você consegue mover o sprite diagonalmente, pressionando por exemplo, seta para esquerda e seta para cima, isto porque o XNA é capaz de detectar mais de uma tecla pressionada simultaneamente.



E que tal adicionarmos uma tecla turbo para o nosso Jogo, de forma que se a tecla espaço for pressionada, ele se movimente 3 vezes mais rápido?

Para executarmos tal façanha, primeiro precisamos mudar a declaração de nossa constante velocidade, tornando-a uma variável:

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

    Texture2D sprite;
    Vector2 vetor;
    int velocidade = 3;
```

O próximo passo é verificar se a tecla espaço esta pressionada, alterando o valor da agora variável velocidade para 9, e em caso negativo, voltando o valor para 3. Veja como ficou o código do nosso evento Update:

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

    //captura o estado do teclado
    KeyboardState teclado = Keyboard.GetState();

    //Turbo
    if (teclado.IsKeyDown(Keys.Space))
        velocidade = 10;
    else
        velocidade = 3;


    //Verifica se alguma tecla de movimento esta pressionada
    if (teclado.IsKeyDown(Keys.Left))
        vetor.X -= velocidade;
    if (teclado.IsKeyDown(Keys.Right))
        vetor.X += velocidade;
    if (teclado.IsKeyDown(Keys.Up))
        vetor.Y -= velocidade;
    if (teclado.IsKeyDown(Keys.Down))
        vetor.Y += velocidade;


    base.Update(gameTime);
}
```

Agora faça um teste!

A classe keybord ainda tem métodos para verificar se a tecla não esta pressionada – IsKeyDown – e GetKeypressed retorna um array com todas as teclas pressionadas no momento da captura.

Adicionando controle com o Mouse

Podemos adicionar movimento ao sprite através do mouse de forma semelhante ao feito com o teclado: Declaramos uma variável do tipo MouseState, capturamos a posição atual do mouse, e atribuímos os valores a nosso vetor. Criamos uma nova solução Mouse, porém você pode optar por apenas alterar o evento Update:

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

    //Define a posição do sprite pelo Mouse
    MouseState mouse = Mouse.GetState();
    vetor.X = mouse.X;
    vetor.Y = mouse.Y;


    base.Update(gameTime);
}
```

Roda a aplicação, movimente o mouse e veja o que nossa nave espacial acompanha o movimento do mouse.

E quanto aos botões do mouse? Podemos inspecionar a mesma variável e identificar se algum botão foi acionado. No exemplo abaixo, adicionamos uma linha de código para que, caso o usuário clique com o botão esquerdo do mouse seja pressionado, a aplicação será encerrada.

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

    //Define a posição do sprite pelo Mouse
    MouseState mouse = Mouse.GetState();
    vetor.X = mouse.X;
    vetor.Y = mouse.Y;

    if (mouse.LeftButton == ButtonState.Pressed)
    this.Exit();

    base.Update(gameTime);

}
```

Faça um teste.

Exibindo o Cursor do Mouse

Por padrão em um jogo XNA o mouse é invisível para o usuário. Para torná-lo visível basta definir a propriedade IsMouseVisible da classe Game como verdadeiro. Você deve colocar o código no seu método Initialize de seu jogo:

```
protected override void Initialize()
{
     // TODO: Add your initialization logic here
     IsMouseVisible = true;
     base.Initialize();
```

```
}
```

Se você executar a aplicação agora verá que seu sprite fica “pendurado” no mouse pela extremidade superior esquerda. Isto faz sentido, pois atribuímos à posição do mouse as coordenadas X e Y de nosso sprite, cujo sistema é baseado no sistema cartesiano, porém com base superior esquerda. Se você precisar centralizar o mouse, basta subtrair da largura e altura do sprite. Veja o código atualizado em nosso método Update:

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

    //Define a posição do sprite pelo Mouse
    MouseState mouse = Mouse.GetState();
    vetor.X = mouse.X - sprite.Width /2;
    vetor.Y = mouse.Y - sprite.Height /2;

    if (mouse.LeftButton == ButtonState.Pressed)
    this.Exit();

    base.Update(gameTime);

}
```

Conclusão

Neste capítulo tivemos um grande avanço: Agora já podemos capturar as entradas do jogador e fazer com que este movimento sprites capturando a entrada do teclado. No próximo capitulo, vamos avançar ainda mais, verificando quando dois objetos do nosso jogo estão em colisão, uma funcionalidade fundamental para qualquer tipo de jogo!