TI24X7

CURSOS ONLINE EM TECNOLOGIA

Curso XNA Desenvolvimento de jogos

Módulo 3

Autor: Fernando Amaral

WWW.TI24X7.COM.BR

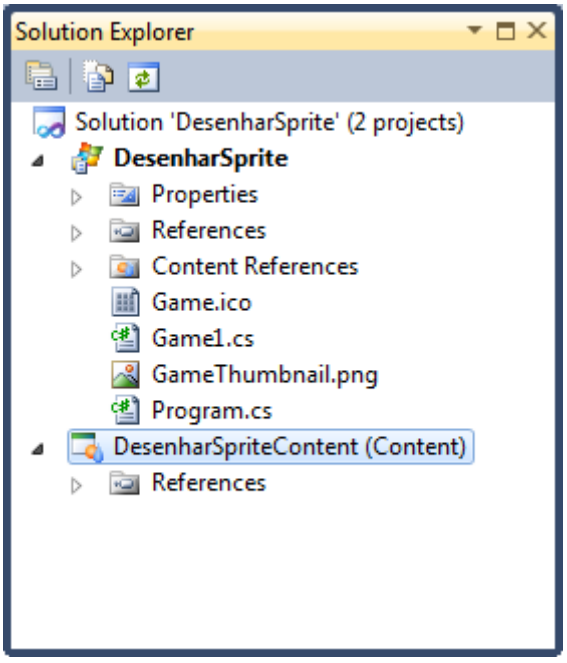
3. Desenhando e Animando Sprites

Neste capítulo a diversão começa. Nossa primeira tarefa será a de desenhar um sprite na tela. Logo em seguida vamos ver como movimentá-lo de forma autônoma, e vamos entender como mante-lo dentro dos limites da tela. Vamos ver ainda como animar um sprite e vamos fechar o capítulo vendo como desenhar texto durante o jogo.

Desenhando Sprites

Abra o Visual Studio e criei um novo projeto, selecionando File, New, Project, Na janela New Project selecione Windows Game (4.0), vamos chamá-lo de DesenharSprite.

Antes de qualquer coisa precisamos de uma imagem para o nosso sprite. No diretório de exemplo deste capítulo você irá encontrar um arquivo de bitmap de nome NaveEspacial.bmp. Lembre-se de qualquer recurso deve ser adicionado ao projeto de conteúdo. Em nossa solução foi criada automaticamente o projeto DesenharSpriteContent, como você pode ver na imagem abaixo:



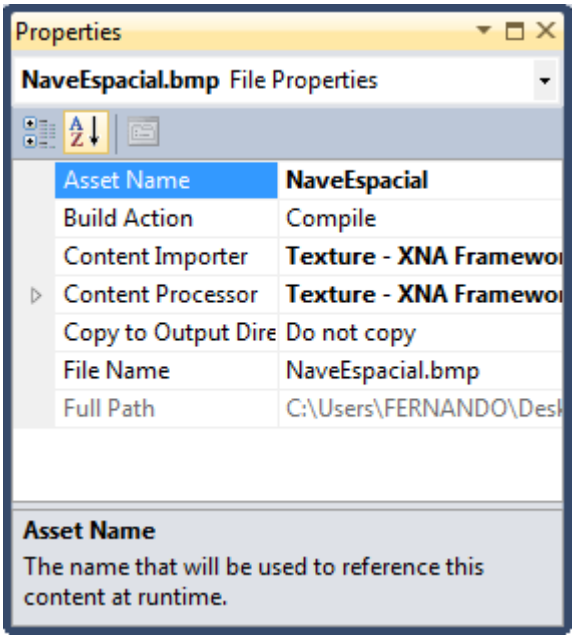
É uma boa prática organizar o conteúdo dentro do projeto de conteúdo, por isso clique com o botão direto sobre o projeto DesenharSpriteContent, selecione Add e New Folder. De o nome da pasta de Imagens. Vamos agora adicionar a imagem. Clique com o botão direito na pasta, selecione Add, Existem Item. Localize o arquivo NaveEspacial.bmp no diretório de exemplo do livro e adicione a solução.

Nota:

Os sprites utilizados neste livro fazem parte da SpriteLib GPL, de autoria de Ari Feldman e

disponibilizados através de licença GPL. Para saber mais, acesse http://www.flyingyogi.com/fun/spritelib.html

Clique sobre a imagem a pouco adicionada para selecioná-la e visualize a janela de propriedades:



Observe a propriedade Asset Name. O valor desta propriedade é a forma como faremos referencia a esta imagem em nosso código, ou seja, o nome do arquivo é irrelevante. O XNA Studio automaticamente define um Asset Name removendo a extensão do nome do arquivo, mas você pode colocar o valor que quiser para esta propriedade. Vamos mante-la com o nome sugerido.

Para desenhar o sprite vamos precisar de uma instancia da classe Texture2D, que suporta uma imagem bidimensional, e do método Draw de uma instancia da classe SpriteBatch. Porém, se você lembra-se do capitulo passado (ou recorde-se agora olhando o código do projeto), o projeto já vem com uma instancia da classe SpriteBatch de nome spriteBatch. Tudo a fazer então é declarar o tipo Texture2D, carregar a imagem, e exibi-la através do método Draw. Vamos ver o procedimento passo a passo.

Primeiro declare a variável sprite, do tipo Texture2D, em nível de classe, o código adicionado esta sublinhado nestes primeiros exemplos esta sublinhado para diferenciar do código gerado pelo Visual Studio:

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

    Texture2D sprite;

    public Game1()
```

Agora localize o método LoadContent. Lembra que neste método devemos carregar todos os nossos recursos? Bom, chegou a hora de carregar nosso sprite:

```
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");

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

Novamente a linha adicionada esta sublinhada. O que fizemos? Carregamos o sprite para nossa variável através do classe Load do gerenciador de conteúdo , processado através do Content Pipeline que estudamos no capitulo anterior. Veja que definimos o tipo do conteúdo, e o Asset Name do mesmo.

Sprite carregado, tudo pronto para desenhá-lo. Se você lembra o capítulo anterior, o método para desenhar na tela é o Draw. O desenho deve ser feito através do método spritebatch, que, como o nome sugere, permite que um lote de sprites sejam desenhados na tela num mesmo frame. O método Draw possuir 7 sobrecargas. A assinatura que vamos utilizar recebe três parâmetros: um tipo Texture2D, onde obviamente informaremos nossa variável sprite, um tipo Vector2, e um tipo Color, que define a cor com que o sprite será pintando. Logo em seguida vamos falar mais sobre o tipo Vector2 e como funciona o sistema de coordenadas em 2D. Semelhante a uma transação de banco de dados, temos que indicar o início e o fim do “lote” de desenhos usando os métodos Begin e End.

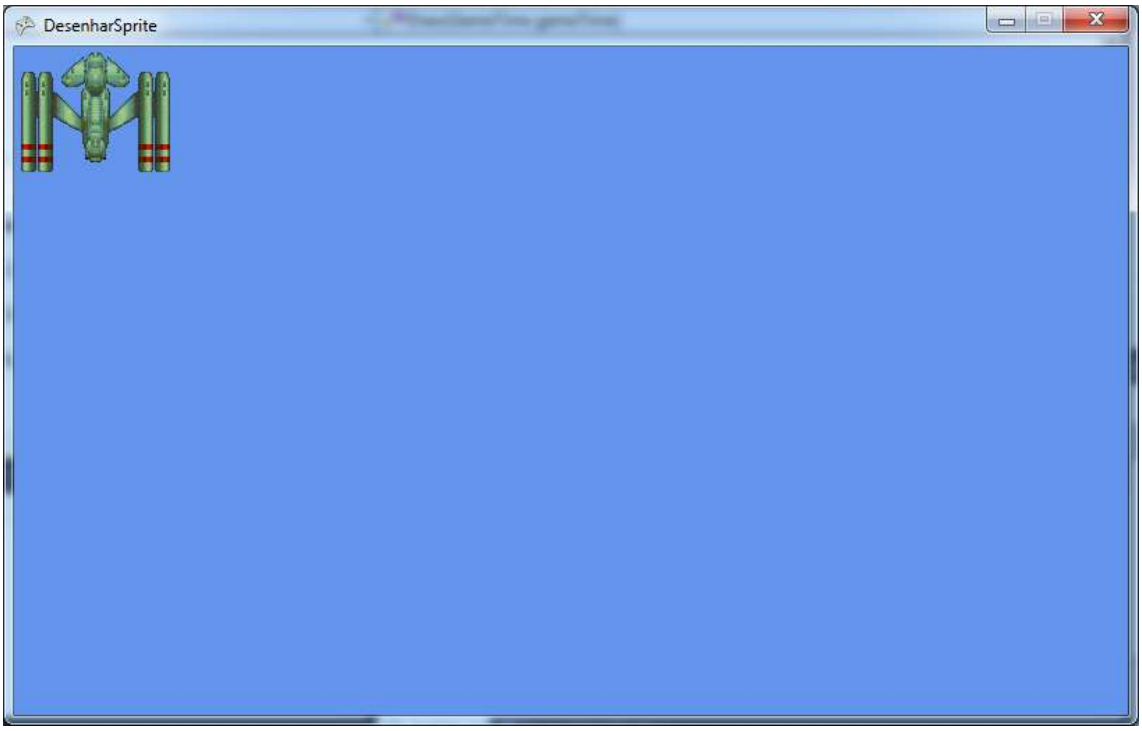
Vamos ao código:

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

     // TODO: Add your drawing code here
             spriteBatch.Begin();
             spriteBatch.Draw(sprite, Vector2.Zero, Color.White);
             spriteBatch.End();
```

```
        base.Draw(gameTime);
    }
```

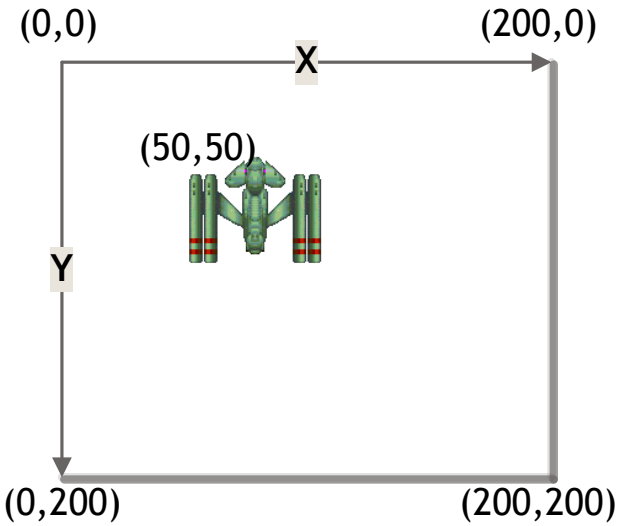
Execute a aplicação. Se tudo deu certo, você verá a tela com na imagem abaixo:



Sistema de Coordenadas em Jogos 2D

Antes de darmos movimento a nosso sprite, precisamos entender como o sistema de coordenas em jogos 2D funciona.

O sistema é semelhante ao sistema de coordenadas cartesianos, sendo o eixo X horizontal, e o eixo Y vertical, também chamados de abscissa e ordenada. A diferença é que no sistema de jogos em 2D, o ponto de coordenadas (0,0) esta localizado na parte superior esquerda do plano, enquanto que no sistema cartesiano, esta localizado também a esquerda, porém na parte inferior. No sistema de Jogos 2D, qualquer valor negativo significa que o objeto estará abandonando os limites da tela. Isto indica que para movermos objetos temos sempre que considerar a altura e a largura da tela, ou seja, a resolução.



Para movermos um objeto horizontalmente, basta incrementarmos o valor de X, da mesma forma, para movermos verticalmente, incrementamos Y. Alterando os dois valores simultaneamente, obtemos um movimento obliquo.

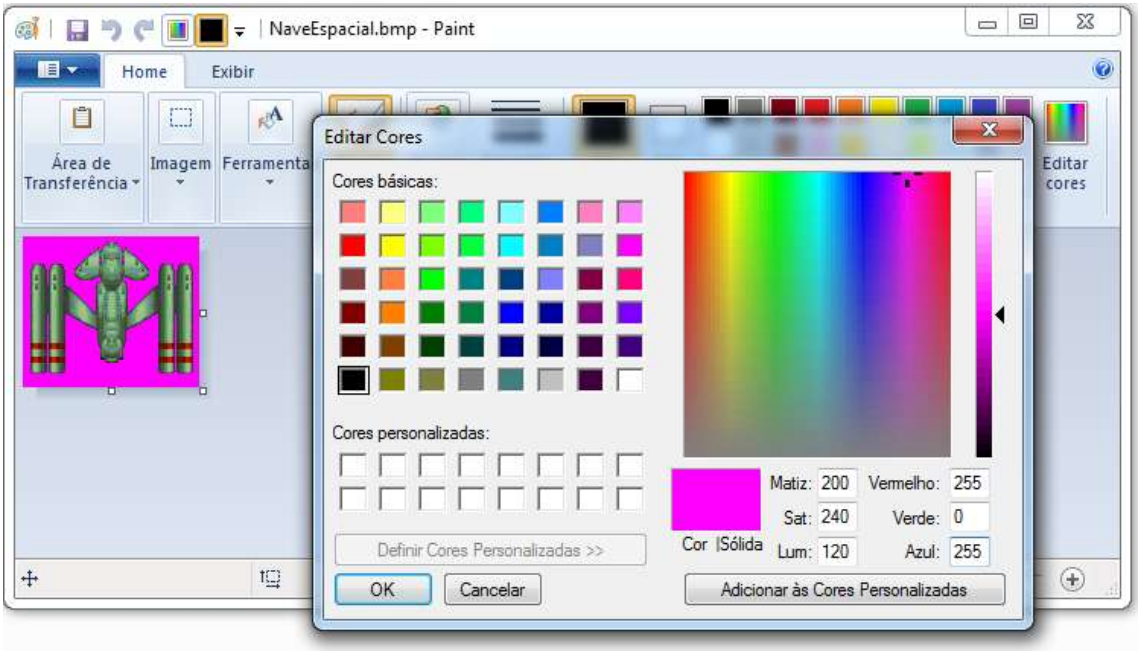
No segundo parâmetro do nosso método Draw, indicamos a posição em que o sprite seria desenhado, passando coordenadas X e Y através do objeto Vector2. Zero informa coordenadas zero, tanto para X quanto para Y.

Considerações sobre transparência

Antes de partirmos para o movimento de nosso sprite, vamos ver algumas considerações importantes sobre transparência. Já que imagens possuem sempre formatos retangulares, precisamos de alguma forma definir áreas transparentes, ou teremos um jogo formando por pilhas de retângulos, o que não seria nada agradável.

Existem duas formas de determinar áreas transparentes em uma imagem. Primeiramente, você pode definir áreas transparentes através do canal alfa de um arquivo do tipo PNG, usando um editor de imagens como o Photoshop ou Gimp.

A outra forma é adicionar a cor Magenta nas áreas em que deseja que haja transparência. Neste caso podemos usar inclusive um bitmap, como o utilizado em nosso exemplo. Se você não precisa de recursos sofisticados de edição de imagens, poderá aplicar o Magenta até com o próprio PaintBrush que acompanha o Windows. Para tal basta abrir a imagem, clicar em editar cores, definir Vermelho como 255, Verde como 0, e Azul 255 (o que equivale a RGB (255,0,255), clique em adicionas as cores personalizadas e em seguida aplique sobre a área que deverá ser apresentada com transparência:



Movendo o Sprite

Mover no Sprite na tela é relativamente simples. Declaramos uma tipo Vector2 em nível de classe, incrementamos os valores de X e/ou Y desta variável no método Update. Depois, é só substituir o segundo parâmetro do método Draw, onde temos Vector2.Zero, pela nossa variável.

Vamos por partes. Primeiro declaramos nossa variável vetor do tipo Vector2 a nível de classe:

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

    Texture2D sprite;
    Vector2 vetor;

    public Game1()
```

Incrementamos X da variável vetor no método Update:

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

    ++vetor.X;

    base.Update(gameTime);
}
```

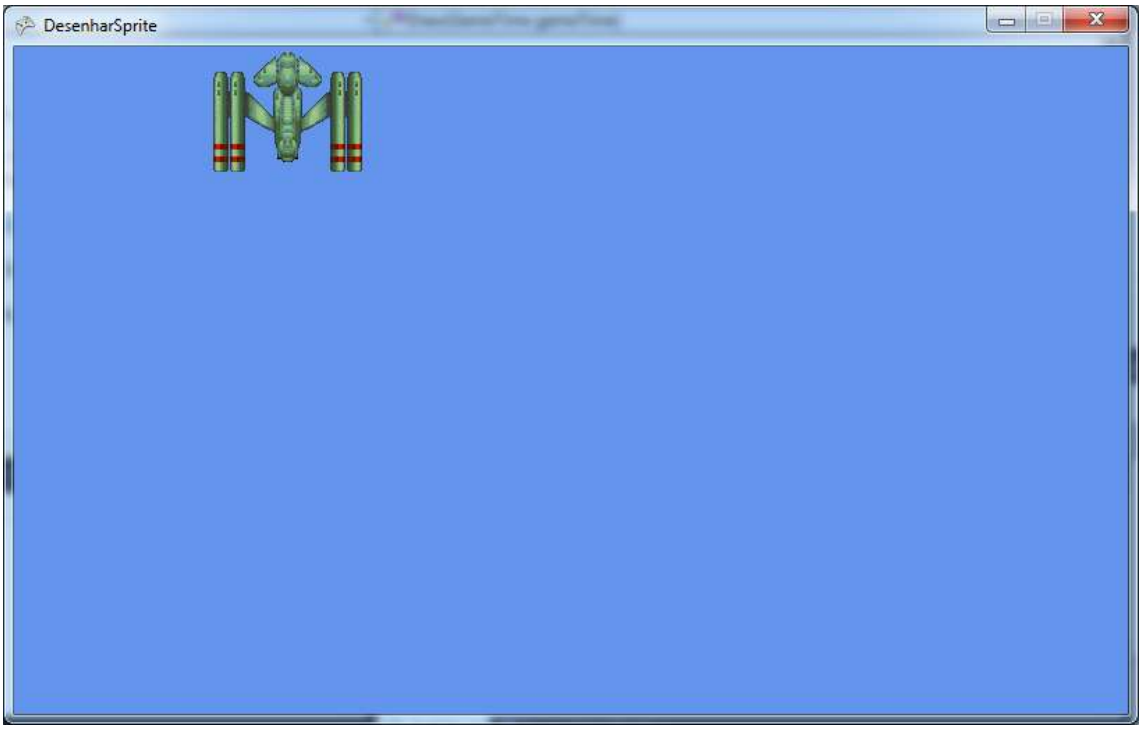
Finalmente, alteramos nosso método Draw:

```
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);
 }
```

Rode a aplicação e veja nossa nave espacial se movimentado verticalmente no topo da tela:



O movimento é apenas vertical porque estamos incrementando apenas o valor de X. Para fazer um movimento obliquo, incrementamos X e Y:

```
++vetor.X;
++vetor.Y;
```

Veja que o objeto se movimenta lentamente, para aumentar a velocidade, basta atribuir um valor superior a X e/ou a Y:

```
vetor.X += 5;
vetor.Y += 5;
```

Respeitando os limites da Tela

Se você acompanhar seu jogo por alguns instantes, vai notar que a nossa nave espacial vai desaparecer no infinito e nunca mais voltar. Ao invés de desaparecer, vamos fazer com a nave ricocheteie nas bordas da janela, assumindo a direção contrária.

Para produzir tal código, devemos verificar se a posição da Nave no eixo X esta maior que a largura da tela – a largura do sprite, ou ainda, se a posição X é menor que zero. Para o eixo Y a lógica é semelhante, verificamos se a posição do sprite esta maior que altura da janela menos a altura do sprite, ou se esta menor que a posição zero.

A largura atual da janela pode ser obtida inspecionado-se a propriedade `Window.ClientBounds.Width`, já para a altura, inspecionamos `Window.ClientBounds.Height`.

Porém como inverter a direção? Existem várias formas. Uma maneira interessante seria criar uma variável auxiliar, velocidade, que determinaria o incremento da posição no eixo X ou Y. Ao intersectar umas das bordas da janela, esta variável seria multiplicada por -1, conseqüentemente a direção da nave espacial mudaria, e o sprite assumiria a direção contrária.

Primeiramente declaramos uma variável velocidade do tipo Vector2 em nível de classe:

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

     Texture2D sprite;
     Vector2 vetor;
     Vector2 velocidade;
```

No método Inicialize inicializamos a velocidade. Usamos o valor 3, ou seja, a sprite deverá se mover a 3 pixels por frame.

```
protected override void Initialize()
{
    velocidade = new Vector2(3,3);

    base.Initialize();
}
```

Finalmente, altere seu método Update para fazer com a imagem ricocheteie, invertendo o incremento da variável velocidade:

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

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

    //atualiza a posição do sprite
    vetor.X += velocidade.X;
    vetor.Y += velocidade.Y;

    base.Update(gameTime);
}
```

Não é preciso nenhuma alteração no método Draw. Rode sua aplicação e observe o resultado, agora sua nave espacial esta “presa” no interior da tela.

Sprites Animados

Evoluímos bastante! Você já entende o sistema de coordenadas em um jogo 2D é já capaz de mover um sprite e de verificar sua intersecção com as bordas da tela. Porém muitas vezes nosso jogo terá um sprite animado: Um disco voador com luzes, uma bomba com o pavio ligando, entre outros. Nesta seção vamos ver como desenhar um sprite animado na tela do jogo.

Embora você possa usar várias imagens para criar um sprite animado, o mais prático – e usual – é usar uma sprite sheet, ou um conjunto de sprites. Um sprite sheet possui as imagens que quando desenhadas na tela do jogo em uma seqüência lógica, darão o efeito a animação. Uma sobrecarga do método Draw da classe spritebach permite que apenas uma área retangular do imagem seja desenhada. O segredo é mudar a área a cada execução do método Draw.

Para isso você precisa saber quantos pixels a imagem tem de largura e de altura, quantas linhas de sprite e quantos sprites por linha. A altura e largura da sprite você pode verificar em tempo de execução, examinando as propriedades width e height da variável Texture2D.

Na imagem que utilizaremos no exemplo, são 512 x 256 pixels, 2 linhas com 8 sprites cada.

Primeiramente, declaramos uma variável sprite do tipo Texture2D e outra variável spriteAtual do tipo Point, que deverá armazenar o sprite que deverá ser desenhado na tela naquele instante:

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

        Texture2D sprite;
        Point spriteAtual = new Point(0, 0);


        public Game1()
```

No método LoadContent carregamos a imagem LavaSaltitante, que você pode encontrar no diretório deste exemplo:

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

        sprite = Content.Load<Texture2D>(@"Imagens\LavaSaltitante");
}
```

No método Update, incrementamos a linha do `spriteAtual` – X – e verificamos se o valor do Frame atual esta maior ou igual que 8, que é o total de sprites que temos em uma linha. Em caso afirmativo, devemos zerar o a coluna do `spriteAtual`, e incrementar a linha, porém se a linha for maior que 2, que o total de numero de linhas, devemos zerá-la.

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

	++spriteAtual.X;
	if (spriteAtual.X >= 8)
	{
		spriteAtual.X = 0;
		++spriteAtual.Y;
		if (spriteAtual.Y >= 2)
			spriteAtual.Y = 0;
	}

}
```

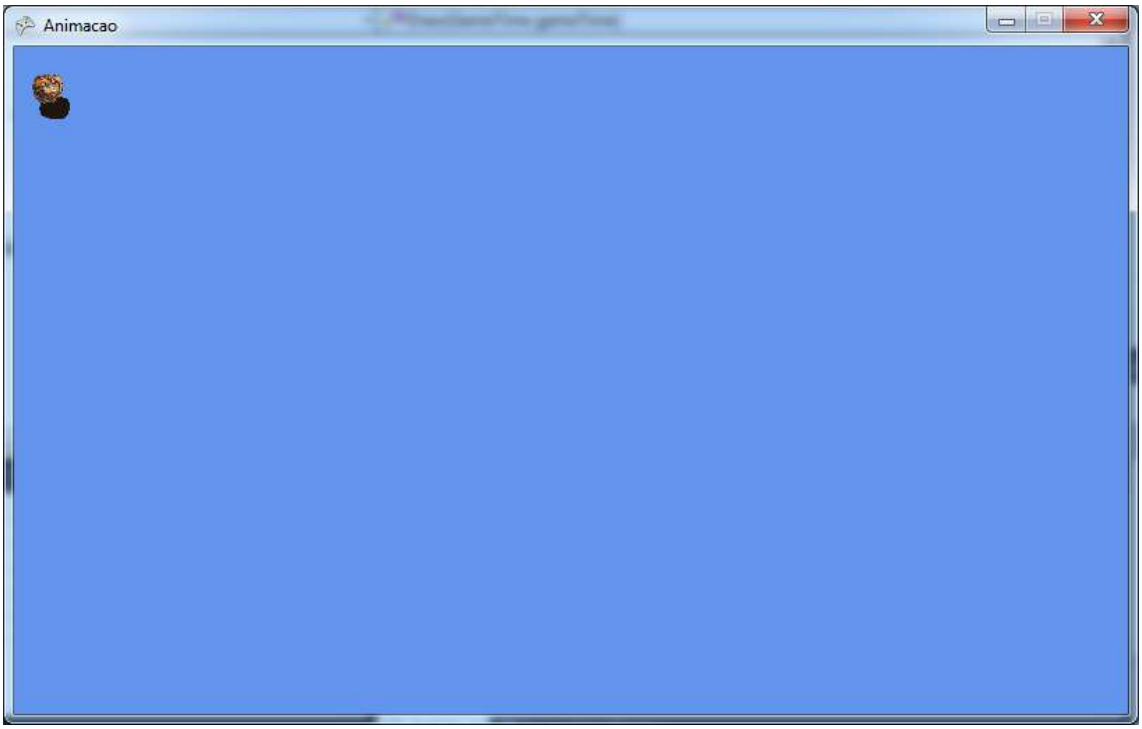
Finalmente desenhamos o sprite, com uma sobrecarga que permite desenhar parte da imagem, selecionada através de um tipo rectangle. Observe que são passados 4 valores, o ponto X e Y de origem do retângulo, e a largura e altura do mesmo.

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

	spriteBatch.Begin();
	spriteBatch.Draw(sprite,Vector2.Zero, new Rectangle(spriteAtual.X *
	     sprite.Width / 8,spriteAtual.Y * sprite.Height / 2,sprite.Width / 8,
	     sprite.Height / 2),Color.White);
	spriteBatch.End();

	base.Draw(gameTime);
}
```

Execute o exemplo, você verá a lava picando no canto superior esquerdo da tela:



Movendo a animação

Agora você já pode facilmente fazer com a sprite se movimente de forma animada na tela, basta usar os conhecimentos das duas últimas seções.

No mesmo projeto, adicione em nível de classe uma variável Vector2 velocidade para determinar a velocidade e Vector2 vetor para guardar o valor da posição do sprite:

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

        Texture2D sprite;
        Point spriteAtual = new Point(0, 0);
        Vector2 vetor;
        Vector2 velocidade;
```

Inicialize a posição do sprite. Neste exemplo vamos movimentá-lo apenas na horizontal, ou seja, no eixo X.

```
protected override void Initialize()
{
        velocidade = new Vector2(3, 0);

        base.Initialize();
}
```

Adicione no método Update, logo abaixo do código incluído na seção anterior. Veja que agora para o sprite bater na lateral, temos que considerar a largura dividido pela quantidade de sprites da imagem, ou seja, 8. Caso contrário o XNA irá considerar a largura de toda a sprite sheet:

```
if (vetor.X > Window.ClientBounds.Width - sprite.Width / 8 ||
             velocidade.X *= -1;

   //atualiza a posição do sprite
   vetor.X += velocidade.X;
```

Finalmente no método Draw, basta trocar o segundo parâmetro para a variável vetor definida em nível de classe.

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

       spriteBatch.Begin();
       spriteBatch.Draw(sprite,vetor, new Rectangle(spriteAtual.X *
                           sprite.Width / 8,spriteAtual.Y * sprite.Height /
                           2,sprite.Width / 8,sprite.Height / 2),Color.White);
       spriteBatch.End();

       base.Draw(gameTime);
}
```

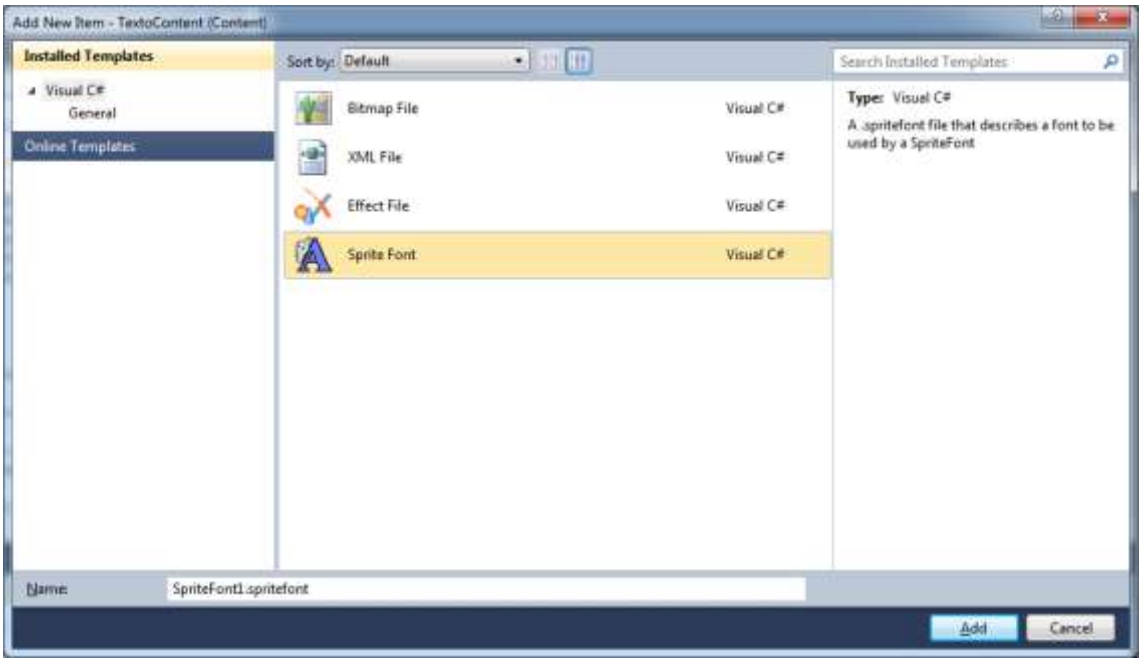
Rode a aplicação, você verá que agora o sprite, além de picar, se movimenta horizontalmente na tela. Estamos evoluindo rapidamente!

Desenhando texto

Durante o jogo geralmente haverá a necessidades de informar o usuário sobre a fase do jogo, os pontos, passar instruções entre outros. Nesta seção vamos ver como desenhar textos. Desenhar texto? Não seria escrever? Em jogos o texto é desenhado na tela como qualquer outro objeto, você verá que inclusive o processo é semelhante.

A primeira etapa é criar um arquivo de fonte de sprite. Este tipo de arquivo não conterá a fonte em si, mas a configuração da mesma. A fonte deverá estar instalada no equipamento onde o jogo estiver rodando. O arquivo de fonte de sprite deve ser adicionado no projeto de recursos, junto com os demais recursos de seu jogo.

Crie um novo projeto de jogo XNA 4.0 do tipo Windows Game, de o nome para o projeto de fontes. No projeto de recursos crie uma pasta fontes. Clique sobre a pasta criada e selecione a opção Add, New Item. Selecione o template Sprite Font, mantenha o nome padrão de SpriteFont1.Sprite.



Abra o arquivo SpriteFont1.Sprite e o examine. Observe que o mesmo é um arquivo XML com alguns nós especiais, que nos permitem definir a fonte e suas características. Por exemplo, para mudar o tipo da fonte, basta alterar o nome no nó FontName:

```
<FontName>Segoe UI Mono</FontName>
```

Por enquanto não altere o arquivo. Abra o arquivo Game1, inicialmente, em nível de classe, declare uma variável fonte, do tipo SpriteFont:

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

     SpriteFont fonte;
```

No método LoadContent carregue a fonte. Note que o processo é semelhante ao carregamento de outros tipos de recursos. Você deve indicar o nome do arquivo Sprite Font criado, sem a extensão, e não o nome da fonte especificado no arquivo:

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

     fonte = Content.Load<SpriteFont>(@"Fontes\SpriteFont1");

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

Finalmente, vamos ao método Draw. Desenhar o texto também é um processo semelhante a desenhar um sprite: Dentro de um batch, usamos ao método DrawString – ao invés de Draw - passando como parâmetros a nossa variável fonte, o texto a ser desenhado, a posição através

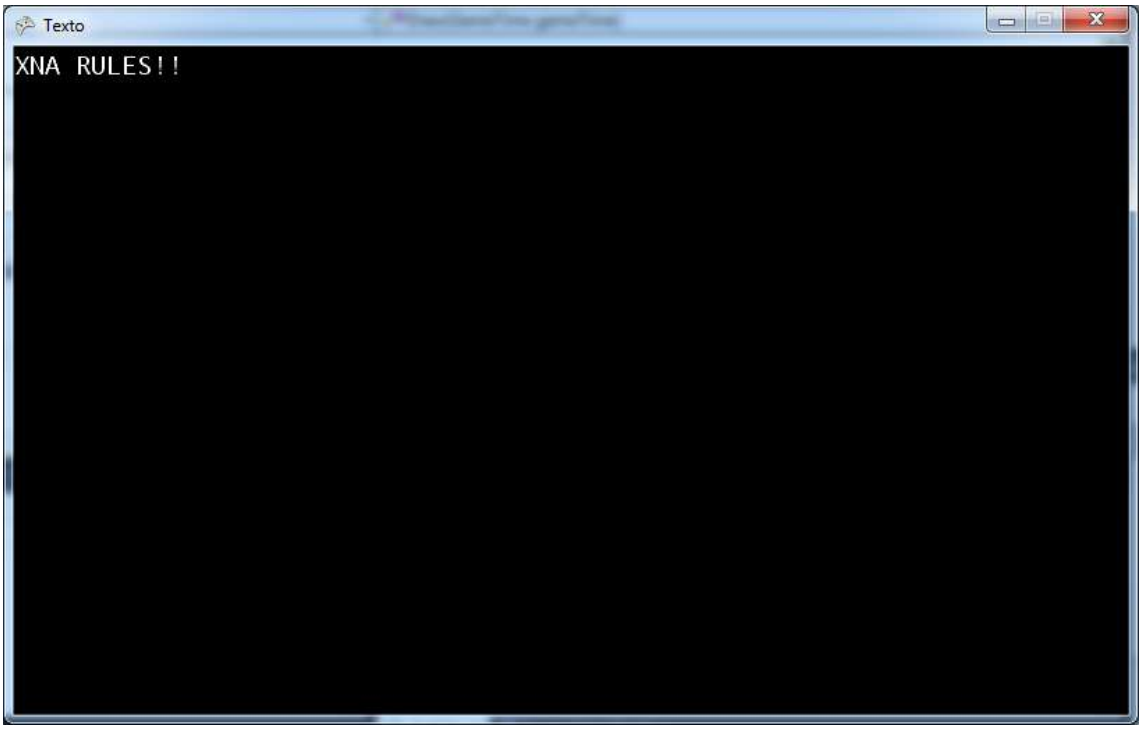
de um objeto Vector2, e finalmente a cor da fonte. Desta vez vamos pintar o fundo da tela de preto para contrastar com nossa tela branca. Observe que no inicio do método Draw, um código incluído pelo XNA Studio em nosso código, chama o método Draw da classe GraphicsDevice. O objetivo é limpar a tela, pintando a mesma com a cor passada como parâmetro. Note que desta vez alteramos o azul padrão pelo preto. Veja o código final:

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

    spriteBatch.Begin();
    spriteBatch.DrawString(fonte,"XNA RULES!!", Vector2.Zero,
             Color.White);
    spriteBatch.End();


    base.Draw(gameTime);
}
```

Rode o exemplo e veja o texto no canto superior esquerdo da tela:



Pronto, agora altere configurações diversas no seu arquivo SpriteFont e rode a aplicação para testar todas as possibilidades de uso de uma fonte.

Mudando a resolução da tela

Você provavelmente vai querer mudar a resolução padrão do jogo XNA. Por exemplo, para iniciar o jogo com uma tela com resolução de 400 x 400, digite o código abaixo no método Initialize:

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    graphics.PreferredBackBufferHeight = 480;
    graphics.PreferredBackBufferWidth = 800;
    graphics.ApplyChanges();


    base.Initialize();
}
```

Rode a aplicação e veja que temos agora o nosso texto desenhado em uma tela quadrada. Mas quanto à tela cheia? Neste caso, após definir a resolução do jogo, atribua o valor verdadeiro a propriedade IsFullScreen de seu dispositivo gráfico:

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    graphics.PreferredBackBufferHeight = 480;
    graphics.PreferredBackBufferWidth = 800;
    graphics.IsFullScreen = true;
    graphics.ApplyChanges();

    base.Initialize();
}
```

Rode a aplicação e pronto! Você tem terá um jogo em tela cheia.

Conclusão

Evoluímos muito! Agora já temos objetos animados andando pela tela do nosso jogo. No próximo capitulo, vamos ver como podemos dar ao usuário o controle sobre um sprite e deixar o jogo interativo!