



TI24X7

CURSOS ONLINE EM TECNOLOGIA

Curso XNA Desenvolvimento de jogos

Módulo 7

Autor: Fernando Amaral

WWW.TI24X7.COM.BR

7. Construindo um Jogo Completo

Chegou a hora de colocar todo o conhecimento adquirido até aqui em prática e construir um jogo completo. Nosso jogo se chamará Astroon, e será um modesto clone do clássico 1945, um jogo no estilo shooter, usando a coleção Sprite Lib que apresentei no capítulo 3. Abaixo você pode ver o jogo, já pronto, em ação:



Planejando o Jogo

Mesmo para um pequeno jogo é necessário fazer um planejamento prévio:





- **O avião do jogador será controlado pelo teclado;**
- **O avião poderá atirar contra os inimigos. Os tiros serão limitados a certa quantidade em um intervalo de tempo;**
- **Haverá dois tipos de inimigos: Lineares e Inteligentes. Os lineares seguirão o caminho em linha reta. Os inteligentes perseguirão o**

avião do jogador, até ultrapassá-lo ou serem destruídos;




- **Haverá contagem de pontos, inimigos lineares terão valor inferior a os inimigos inteligente;**
- **Haverá uma contagem de vidas. O jogador terá direito a 3 vidas;**
- **Os inimigos não deverão atirar contra o avião do jogador. A dificuldade estará em desviar de aviões que não se consiga destruir, dada a limitação no número de tiros.**

Seleção de Sprites

Podem-se usar duas abordagens diferentes no uso de sprites em um jogo. Podemos colocar todos os sprites usados no jogo em uma única sprite sheet. Desta forma, para cada elemento de nosso jogo devemos selecionar a área retangular equivalente a imagem. A outra abordagem é separar sprites em arquivos diferentes, deixando em um mesmo arquivo apenas sprites animados. Eu particularmente prefiro a segunda abordagem: a implementação fica mais fácil e o código fica mais legível. Vamos usar os seguintes sprites em nosso jogo:

	Avião do jogador
	Explosão de um inimigo
	Inimigo Linear
	Inimigo Inteligente

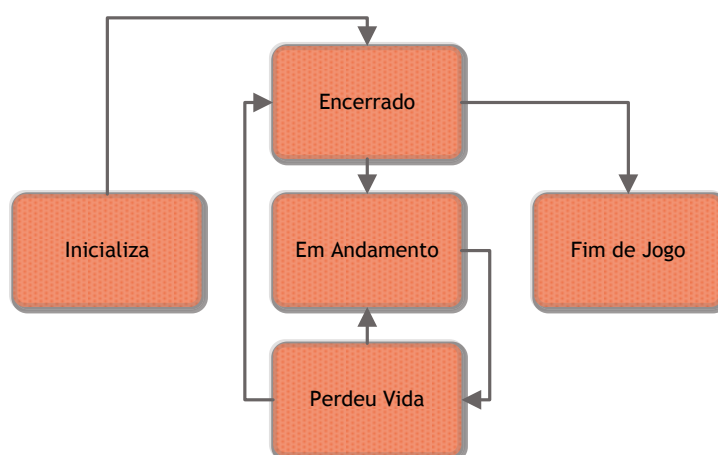
Curso de XNA Desenvolvimento de jogos – Módulo 7

	Splash Screen (Tela de Abertura)
	Tiro
	Controle de Vidas

Controlando o Estado do Jogo

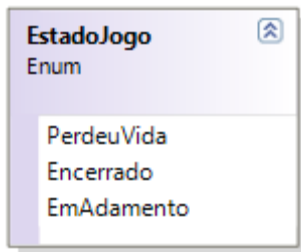
Não queremos fazer um jogo que quando o usuário executa o aplicativo e ele já sai jogando sem o jogador ter tempo de ajustar os dedos no teclado. Jogos em geral têm estados, que são situações em que há um comportamento diferenciado. Exemplos de estado de jogo podem ser pausado, finalizado, trocando de fase, tela inicial, perda de vida entre diversos outros. Os estados de um jogo são importantes para o usuário, para entender o que está acontecendo, e para o próprio código do jogo, pois é preciso saber quando é hora de determinar certos blocos de código.

Nosso jogo terá três estados: Encerrado, Em andamento e perdeu vida. Encerrado é o estado em que o jogo não está em andamento: quando o usuário abre a jogo pela primeira vez ou perde todas as suas vidas, ele estará no estado encerrado. Em andamento é quando, (surpresa!), o jogador está jogando! Perdeu vida é o estado em o avião do jogador foi destruído e o jogo se prepara para reiniciar, se ainda restarem vidas, ou é encerrado, caso não haja mais vidas. O jogo entrará automaticamente nos estados Perdeu Vida e Encerrado. O estado Em andamento será controlado disparado pelo usuário.



Curso de XNA Desenvolvimento de jogos – Módulo 7

Geralmente o estado do jogo é controlado através de uma estrutura enum (enumerador) do C#. Nosso jogo não será diferente: Abaixo o diagrama de nosso enumerador. Mais adiante veremos a codificação do mesmo.



Preparando a Solução

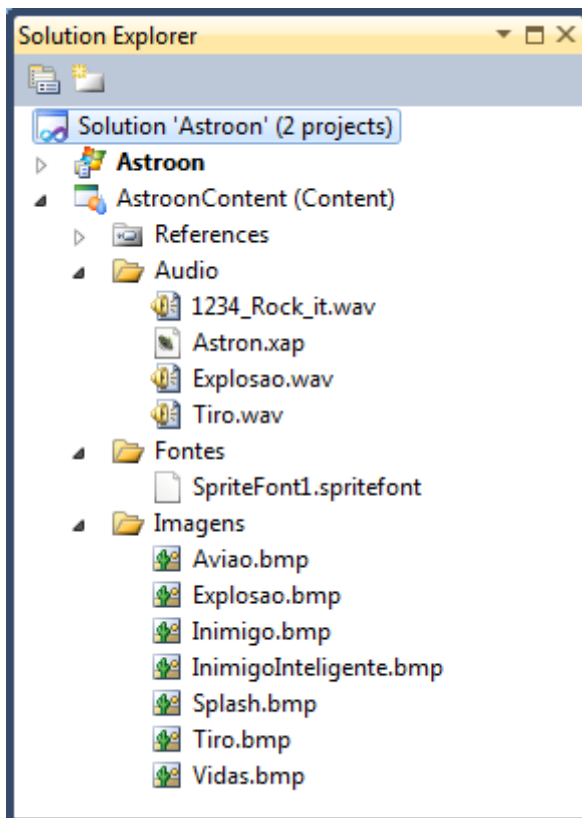
Crie um novo projeto de jogo Windows Game. No projeto de conteúdo, adicione as pastas Imagens, Audio, e Fontes.

Crie um projeto de áudio com o XACT utilizando os arquivos 1234_Rock_it.wav, Explosao.wav e Tiro.wav. Você encontra os arquivos junto aos códigos fontes que acompanham este livro. Marque o arquivo 1234_Rock_it.wav para loop infinito. Salve o projeto com o nome Astron.xap na pasta Audio criada anteriormente. Adicione o arquivo Astron.xap a pasta Audio do projeto de recursos.

Na pasta imagens, adicione as imagens Aviao.bmp, Explosao.bmp, Inimigo.bmp, InimigoInteligente.bmp, Splash.bmp, Tiro.bmp e Vidas.bmp. Você também encontra os arquivos junto aos códigos fontes que acompanham este livro.

Finalmente, na pasta Fontes adicione uma Sprite Font. Mantenha o nome padrão. Localize o nó Size no XML e altere para 24. Abaixo você pode visualizar como deverá ficar seu projeto de recursos. Note que os arquivos de áudio foram incluídos na solução. Isto foi feito apenas para controle, pois quando você inclui o arquivo do projeto criado no XACT, os arquivos originais de áudio não precisam ser incluídos no projeto.

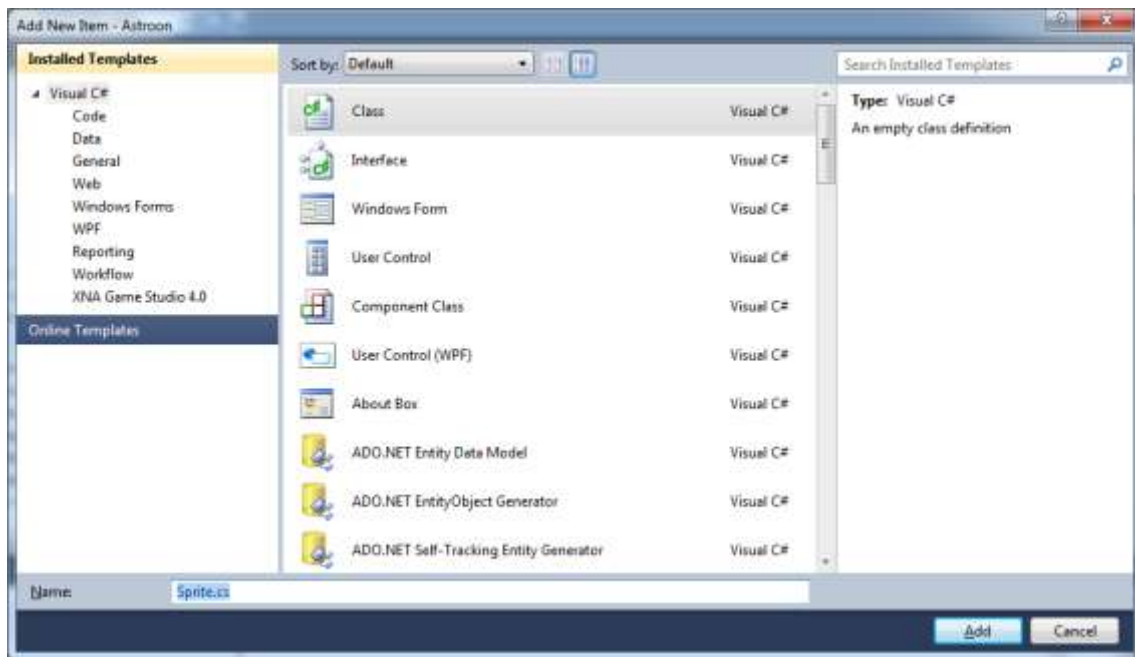
Curso de XNA Desenvolvimento de jogos – Módulo 7



Criando uma Classe Sprite

Nos capítulos anteriores buscamos manter ao máximo a simplicidade do código, neste capítulo vamos utilizar alguns conceitos fundamentais de Orientação a Objetos. Todos os bitmaps adicionados a nossa pasta imagens no projeto de conteúdos serão sprites em nosso jogo. Todos terão métodos e propriedades comuns. Alguns existirão de forma única, outros como explosões e inimigos, poderão existir em vários ao mesmo tempo. A melhor forma de controlar nossos sprites é claro, construindo uma classe Sprite. Este será o elemento principal de nosso jogo e você encontrará na internet dezenas de implementações diferentes. A nossa classe procurou manter a simplicidade, implementando as funções necessárias para atender os requisitos descritos anteriormente para nosso jogo. Para criar uma classe Sprite em seu projeto, clique com o botão direito sobre o projeto Astroon, selecione Add, New Item, selecione o template Class, de o nome de Sprite.cs e clique em Add:

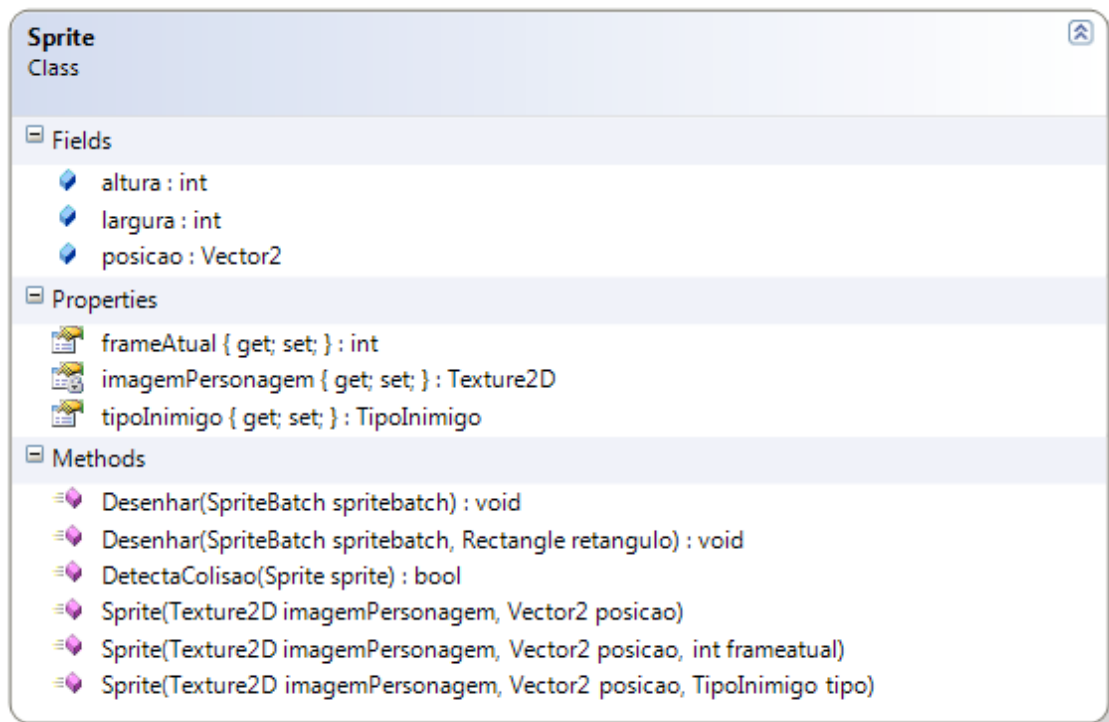
Curso de XNA Desenvolvimento de jogos – Módulo 7



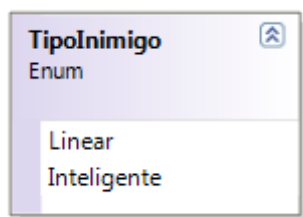
Antes da codificação, vamos examinar um diagrama de nossa classe produzido pelo Visual Studio:

A classe é composta dos campos altura e largura e posição. Temos ainda as propriedades frameAtual, imagemPersonagem e tipoInimigo. Os métodos implementados são: um construtor com mais duas sobrecargas, um método para detectar colisões e um método Desenhar com mais uma sobrecarga:

Curso de XNA Desenvolvimento de jogos – Módulo 7



O campo `tipoInimigo` é do tipo `TipoInimigo`. Lembre-se no planejamento de nosso jogo que poderíamos ter inimigos lineares e inteligentes:



Vamos agora analisar o código da classe. Para facilitar a compreensão, faremos a análise em blocos, ao final da seção, estará toda a implementação da classe.

Codificando a Classe Sprite

Nossa classe inicia com a declaração do enumerador `TipoInimigo`, que poderão ter os valores `Linear` e `Inteligente`. Por padrão o C# atribui os valores 0 e 1 para os membros do enumerador, porém, para fins didáticos, deixamos os valores explícitos. Temos em seguida o início da estrutura de nossa classe, com a declaração de três campos: `Vector2`, para a posição do Sprite, `largura` e `altura`, para o tamanho do Sprite. Veja que `altura` e `largura` são somente leitura, isto porque um objeto sprite – ou seja, uma instancia da classe `sprite` – não poderá alterar a `altura` e `largura` da imagem. Temos em seguida as propriedades `tipoInimigo`, que implementa o enumerador `TipoInimigo`, `imagemSprite` e `frameAtual`. Observe que `ImagemSprite` tem a visibilidade privada, isto porque o `Sprite` será desenhado através do método `draw` de nossa

Curso de XNA Desenvolvimento de jogos – Módulo 7

classe, o bitmap não será acessado por objetos do tipo Sprite. FrameAtual é para controlar SpriteAnimados. No nosso jogo, teremos apenas explosões como sprite animado.

```
namespace Astroon
{
    //Enumerador para o tipo do inimigo
    public enum TipoInimigo
    {
        Linear =0,
        Inteligente=1
    }

    class Sprite
    {
        //Posicao
        public Vector2 posicao;
        //Largura do Srite
        public readonly int largura;
        //Altura do Sprite
        public readonly int altura;
        //Tipo do inimigo
        public TipoInimigo tipoInimigo { get; set; }
        //Image
        private Texture2D imagemSprite;
        //Frame atual, para sprite animado
        public int frameAtual { get; set; }
    }
}
```

Nossa classe conta com o construtor sobrecarregado três vezes. O primeiro construtor recebe a imagem do personagem e sua posição inicial. A segunda sobrecarga recebe o enumerador TipoInimigo, e será usado para criar inimigos para nosso jogo. A terceira sobrecarga recebe o FrameAtual, a fim de controlar Sprites animados, como explosões e vidas.

```
//Construtor
public Sprite(Texture2D imagemPersonagem, Vector2 posicao)
{
    this.imagemSprite = imagemPersonagem;
    this.posicao = posicao;
    this.altura = this.imagemSprite.Height;
    this.largura = this.imagemSprite.Width;
}

//Sobrecargado construtor, com tipo de inimigo
public Sprite(Texture2D imagemPersonagem, Vector2 posicao, TipoInimigo
    tipo)
{
    this.imagemSprite = imagemPersonagem;
    this.posicao = posicao;
    this.tipoInimigo = tipo;
    this.altura = this.imagemSprite.Height;
    this.largura = this.imagemSprite.Width;
}

//Sobrecarga do construtor, com controle de Frame
public Sprite(Texture2D imagemPersonagem, Vector2 posicao, int
    frameAtual)
```

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
{
    this.imagemSprite = imagemPersonagem;
    this.posicao = posicao;
    this.frameAtual = frameAtual;
    this.altura = this.imagemSprite.Height;
    this.largura = this.imagemSprite.Width;
}
```

A seguir nossa classe implementa o método Desenhar, que desenha o Sprite do objeto. Note que o método é bem simples: recebe como parâmetro o Spritebatch atual do jogo. Uma sobrecarga recebe um tipo retângulo, para desenharmos parte da imagem. Usaremos esta sobrecarga para desenharmos a explosão e as vidas:

```
//Desenha o sprite
public void Desenhar(SpriteBatch spriteBatch)
{
    spriteBatch.Draw(this.imagemSprite, this.posicao, Color.White);
}

//Sobrecarga do método desenhar, definindo retângulo
public void Desenhar(SpriteBatch spriteBatch, Rectangle retangulo)
{
    spriteBatch.Draw(this.imagemSprite, this.posicao, retangulo,
        Color.White);
}
```

O último método de nossa classe é a detecção de colisão. O método é bem semelhante ao que estudamos no capítulo 5, porém, ele recebe como parâmetro outro objeto Sprite, com o qual a colisão será testada. Este método será usado em nosso jogo para testar colisões entre tiros e inimigos e inimigos e a nave do jogador.

```
//Detecta colisão
public bool DetectaColisao(Sprite sprite)
{
    return (this.posicao.X + this.imagemSprite.Width > sprite.posicao.X &&
        this.posicao.X < sprite.posicao.X + sprite.imagemSprite.Width &&
        this.posicao.Y + this.imagemSprite.Height > sprite.posicao.Y &&
        this.posicao.Y < sprite.posicao.Y + sprite.imagemSprite.Width);
}
```

Classe Sprite Completa

Abaixo você pode ver a implementação completa da nossa classe Sprite:

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
namespace Astroon
{
    //Enumerador para o tipo do inimigo
    public enum TipoInimigo
    {
        Linear =0,
        Inteligente=1
    }

    class Sprite
    {
        //Posicao
        public Vector2 posicao;
        //Largura do Srite
        public readonly int largura;
        //Altura do Sprite
        public readonly int altura;
        //Tipo do inimigo
        public TipoInimigo tipoInimigo { get; set; }
        //Image
        private Texture2D imagemSprite { get; set; }
        //Frame atual, para sprite animado
        public int frameAtual { get; set; }

        //Construtor
        public Sprite(Texture2D imagemPersonagem, Vector2 posicao)
        {
            this.imagemSprite = imagemPersonagem;
            this.posicao = posicao;
            this.altura = this.imagemSprite.Height;
            this.largura = this.imagemSprite.Width;
        }

        //Sobrecargado construtor, com tipo de inimigo
        public Sprite(Texture2D imagemPersonagem, Vector2 posicao, TipoInimigo
            tipo)
        {
            this.imagemSprite = imagemPersonagem;
            this.posicao = posicao;
            this.tipoInimigo = tipo;
            this.altura = this.imagemSprite.Height;
            this.largura = this.imagemSprite.Width;
        }

        //Sobrecarga do contrutor, com controle de Frame
        public Sprite(Texture2D imagemPersonagem, Vector2 posicao, int
```

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
        frameAtual)
    {
        this.imagemSprite = imagemPersonagem;
        this.posicao = posicao;
        this.frameAtual = frameAtual;
        this.altura = this.imagemSprite.Height;
        this.largura = this.imagemSprite.Width;
    }

    //Desenha o sprite
    public void Desenhar(SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(this.imagemSprite, this.posicao, Color.White);
    }

    //Sobrecarga do método desenhar, definindo retângulo
    public void Desenhar(SpriteBatch spriteBatch, Rectangle retangulo)
    {
        spriteBatch.Draw(this.imagemSprite, this.posicao, retangulo,
            Color.White);
    }

    //Detecta colisão
    public bool DetectaColisao(Sprite sprite)
    {
        if (this.posicao.X + this.imagemSprite.Width > sprite.posicao.X &&
            this.posicao.X < sprite.posicao.X + sprite.imagemSprite.Width &&
            this.posicao.Y + this.imagemSprite.Height > sprite.posicao.Y &&
            this.posicao.Y < sprite.posicao.Y + sprite.imagemSprite.Width)
            return true;

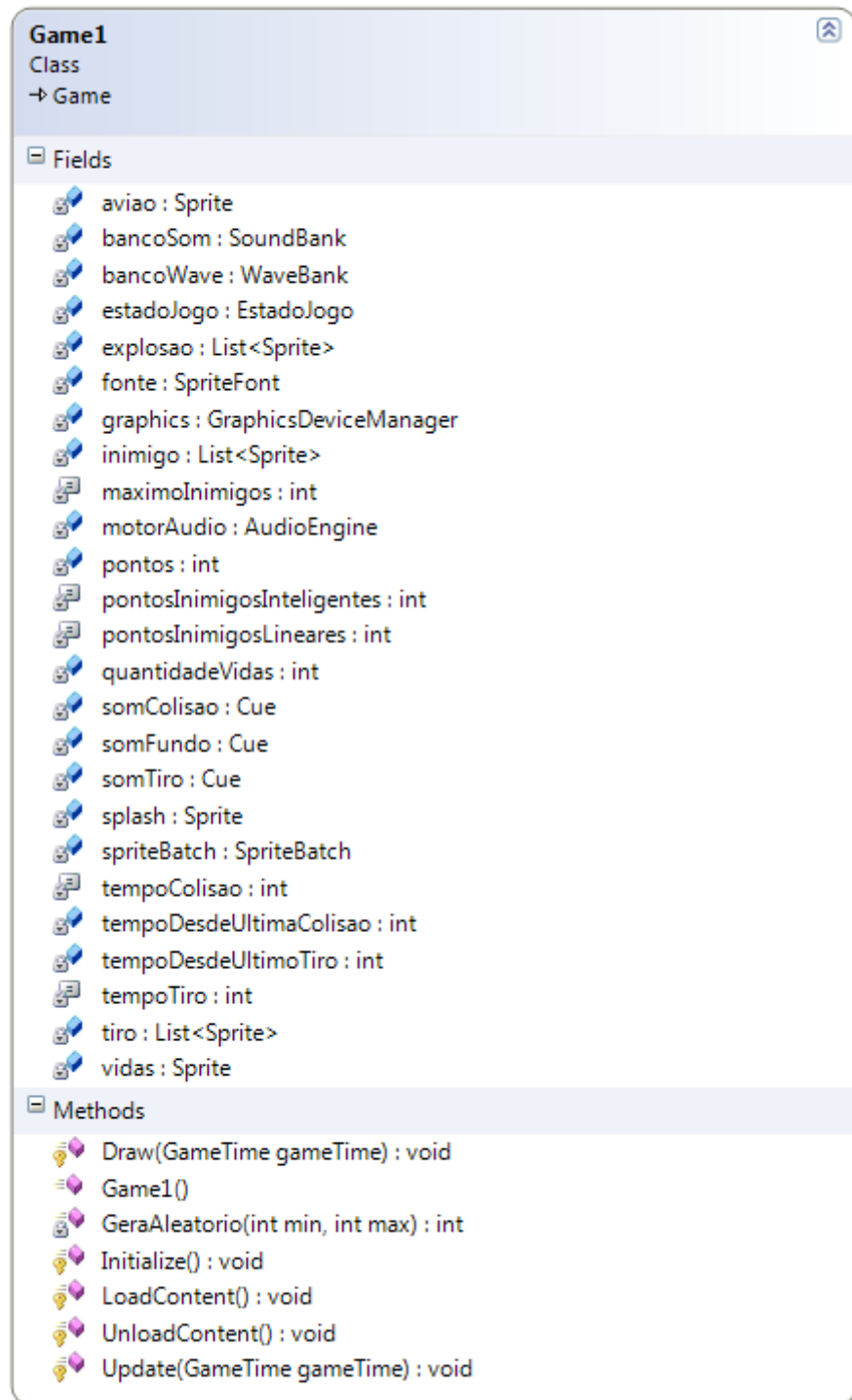
        else
            return false;
    }
}
}
```

Construindo o Jogo:

O codificação principal de nosso jogo será feito no arquivo Game1.cs já existente e girará em torno de instanciar e manipular objetos do tipo Sprite. Vamos analisar o código passo a passo, no final, novamente, apresentaremos a implantação completa da classe.

Inicialmente analisamos o diagrama do classe Game:

Curso de XNA Desenvolvimento de jogos – Módulo 7



Observe que temos duas dezenas de campos e alguns poucos métodos. O único método que não foi criado automaticamente pelo XNA e que não faz parte da estrutura inicial do jogo, é o `GeraAleatorio`. Falaremos mais adiante dele.

Primeiramente nosso enumerador para controlar o estado do jogo:

```
//Enumerados para verificação do estado do jogo
```

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
enum EstadoJogo
{
    PerdeuVida,
    Encerrado,
    EmAdamento
}
```

Vamos ver os campos criados, comentado a função de cada um:

Temos inicialmente um campo para controlar o estado atual do jogo:

```
//Variável para manter o estado do jogo
private EstadoJogo estadoJogo;
```

Agora declaramos diversos tipos: sprites, aviao, vidas, splash. Em seguida temos inimigo, tiro e explosao. Observe que estes três objetos foram declarados na forma de listas genéricas, porque, ao contrário do avião controlado pelo jogador que será único, podemos ter muitos inimigos, tiros e explosões simultaneamente no nosso jogo:

```
//Avião do Jogador
private Sprite aviao;
//Desenhas vidas restantes na tela
private Sprite vidas;
//Tela de Splash (Abertura)
private Sprite splash;
//Lista de Inimigos
private List<Sprite> inimigo = new List<Sprite>();
//Lista de Tiros
private List<Sprite> tiro = new List<Sprite>();
//Lista de Explosões
private List<Sprite> explosao = new List<Sprite>();
//Fonte para escrever na tela
private SpriteFont fonte;
```

Temos agora a declaração dos objetos de áudio que serão usados no jogo. Basicamente termos três tipos de som: som de fundo, que será executado permanentemente durante o jogo, som de colisão e som quando os tiros são disparados.

```
//Variáveis de Audio
private AudioEngine motorAudio;
private WaveBank bancoWave;
private SoundBank bancoSom;
private Cue somFundo;
private Cue somColisao;
private Cue somTiro;
```

Finalmente, temos um conjunto de variáveis para controles diversos de nosso jogo. Uma variável para contagem de pontos, uma para a quantidade de vidas restantes que é inicializada com 3, que é a quantidade de vidas que nosso jogador terá ao iniciar o jogo. Em seguida temos duas variáveis para controlar o tempo de tiro do jogador, pois se ele puder atirar de forma indeterminada o jogo ficará muito fácil. TempoDesdeUltimoTiro é uma variável de controle, e tempoTiro é o tempo em milissegundos que o jogador terá que aguardar antes de um novo tiro. Você poderá mudar a dificuldade do jogo alterando o valor desta constante.

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
//Pontos
private int pontos = 0;
//Quantidade de vidas
private int quantidadeVidas = 3;
//Controla o tempo do tiro
private int tempoDesdeUltimoTiro = 0;
//Aqui altera a dificuldade do jogo
private const int tempoTiro = 200;
```

Em seguida temos os campos para controlar o tempo de intervalo entre as colisões da nave espacial do usuário, para que o jogo possa se reorganizar e o jogador se recompor. tempoDesdeUltimaColisao é uma variável de controle, e tempoColisao é uma constante que vai controlar este intervalo entre uma colisão e outra. A constante maximoInimigos é o controle do jogo para o número Máximo de inimigos que deverão existir na tela, no futuro você poderá facilmente transformá-la numa variável a aumentar a quantidade de inimigos a medida que a dificuldade do jogo aumenta. Finalmente, temos duas constantes para indicar os pontos que o jogador vai ganhar destruindo os inimigos, como teremos dois tipos de inimigos, o jogador poderá pontuar de duas formas diferentes:

```
//Conta tempo para retorno ao jogo após a colisão do avião
private int tempoDesdeUltimaColisao = 0;
private const int tempoColisao = 1000;
//Número máximo de inimigos presentes na tela, altera a dificuldade do jogo
private const int maximoInimigos = 10;
//Pontos ganhos com a destruição de inimigos inimigos
private const int pontosInimigosLineares = 10;
private const int pontosInimigosInteligentes = 50;
```

Agora vamos estudar o método Load. Não há nada de novo aqui, os objetos que serão usados durante o jogo são carregados: O avião do jogador, vidas, splash, fonte, variáveis de áudio, tocamos o som de fundo e definimos o estado inicial do jogo: Encerrado. Veja que não faz sentido aqui carregar inimigos, explosões, tiros ou sons de tiros e de explosões: faremos isto dinamicamente durante o jogo, aqui, carregamos apenas os objetos que serão usados do início ao fim do jogo:

```
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);

    //Adiciona o avião que será controlado pelo usuario
    aviao = new Sprite(Content.Load<Texture2D>(@"Imagens\Aviao"), new
        Vector2(Window.ClientBounds.Width / 2, Window.ClientBounds.Height /
        2));

    //Carrega imagem que mostra a quantidade de vidas
    vidas = new Sprite(Content.Load<Texture2D>(@"Imagens\Vidas"), new
        Vector2(0,0));

    //Carrega imagem da tela de Splash
    splash = new Sprite(Content.Load<Texture2D>(@"Imagens\Splash"), new
        Vector2(0, 0));
    splash.posicao.X = Window.ClientBounds.Width / 2 - splash.largura / 2;
    splash.posicao.Y = Window.ClientBounds.Height / 2 - splash.altura / 2;

    //Carrega a fonte do Jogo
    fonte = Content.Load<SpriteFont>(@"Fontes\SpriteFont1");
```

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
//Variáveis de som
motorAudio = new AudioEngine(@"Content\Audio\Astron.xgs");
bancoWave = new WaveBank(motorAudio, @"Content\Audio\Wave Bank.xwb");
bancoSom = new SoundBank(motorAudio, @"Content\Audio\Sound Bank.xsb");

//Inicia Som de fundo
somFundo = bancoSom.GetCue("1234_Rock_it");
somFundo.Play();

//Estado inicial do jogo
estadoJogo = EstadoJogo.Encerrado;
}
```

Agora vamos ao nosso método Update. Iniciamos o bloco colocando o código que é independente do estado do nosso Jogo: Primeiro capturamos o estado do teclado e permitimos que o jogo seja encerrado a qualquer momento pressionando-se a tecla Escape:

```
//Captura o estado do teclado
KeyboardState estadoTeclado = Keyboard.GetState();

//Na tecla esc sai do jogo
if (estadoTeclado.IsKeyDown(Keys.Escape))
    this.Exit();
```

Antes de seguirmos vamos a algumas reflexões. O método terá atuações diferentes conforme o estado do jogo, definido pela nossa variável estadoJogo, que é uma instância da estrutura EstadoJogo. Uma boa forma de organizar o código para os diferentes estados de nosso jogo é criar uma estrutura C# switch, codificando dentro de cada bloco o código equivalente ao estado:

```
switch (estadoJogo)
{
    case EstadoJogo.PerdeuVida:
        //Aqui colocamos o código para ser executando quando o
        //usuário perde uma vida
        break;
    case EstadoJogo.Encerrado:
        //Aqui colocamos o código para ser executando quando o jogo
        //esta encerrado
        break;
    case EstadoJogo.EmAndamento:
        //Aqui colocamos o código para ser executando quando o jogo
        //esta em andamento
        break;
}
```

Veja por exemplo que quando o jogo esta encerrado, não há necessidade de detectar colisões, criar explosões ou somar pontos, desta forma a lógica do código ficará mais clara. Vamos estudar agora cada bloco da nossa estrutura switch. Como você deve imaginar, a emoção estará toda concentrada no estado EmAndamento

Update: PerdeuVida

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
switch (estadoJogo)
{
    case EstadoJogo.PerdeuVida:
        break;
    case EstadoJogo.Encerrado:
        break;
    case EstadoJogo.EmAndamento:
        break;
    default:
        break;
}
```



Quando o usuário perde uma vida, basicamente é necessário apenas contar o tempo para o retorno ao jogo. Para controlar o tempo usamos a variável `tempoDesdeUltimaColisao`, incrementando a mesma com os milissegundos percorridos do jogo, até ficar maior que a constante `tempoColisao`. Neste momento por segurança, decrementamos `tempoDesdeUltimaColisao` com o valor de `tempoColisao`, e passamos o estado do jogo para `EmAndamento`. Talvez você se pergunte, mas se o estado é de perda de vida do jogador, aonde eu decremento a quantidade de vidas e verifico se o jogo não acabou? Isto na verdade será realizado no estado `EmAndamento`, lá, após detectar uma colisão do avião do jogador com um inimigo, as vidas são decrementadas e o estado do jogo é alterado para `PerdeuVida` se ainda restarem vidas ou `Encerrado` caso não hajam mais vidas.

```
//aguarda antes de mudar o estado do jogo
tempoDesdeUltimaColisao += gameTime.ElapsedGameTime.Milliseconds;
if (tempoDesdeUltimaColisao > tempoColisao)
{
    tempoDesdeUltimaColisao -= tempoColisao;
    estadoJogo = EstadoJogo.EmAndamento;
}
}
```

Update: Encerrado

```
switch (estadoJogo)
{
    case EstadoJogo.PerdeuVida:
        break;
    case EstadoJogo.Encerrado:
        break;
    case EstadoJogo.EmAndamento:
        break;
    default:
        break;
}
```



Para o caso de jogo encerrado, tudo a fazer é colocar um código permitindo que o jogador reinicie o jogo tecando F2, caso em que o estado do jogo muda para `EmAndamento`. Reiniciar o jogo requer algumas ações, como zerar os pontos e reiniciar as vidas em 3. Isto também

Curso de XNA Desenvolvimento de jogos – Módulo 7

deverá ser feito no estado EmAndamento, quando, após detectada a colisão, percebermos que não restam mais vidas ao jogador

```
//Pressionando F2 o jogo é reiniciado
if (estadoTeclado.IsKeyDown(Keys.F2))
{
    estadoJogo = EstadoJogo.EmAndamento;
}
```

Update: EmAndamento

```
switch (estadoJogo)
{
    case EstadoJogo.PerdeuVida:
        break;
    case EstadoJogo.Encerrado:
        break;
    case EstadoJogo.EmAndamento:
        break;
    default:
        break;
}
```



Para o jogo em andamento, o primeiro trecho de código permite o movimento do avião pelo teclado, mantendo-o dentro dos limites da tela:

```
//movimento do aviao
if (estadoTeclado.IsKeyDown(Keys.Right) && aviao.posicao.X <
    Window.ClientBounds.Width - aviao.largura)
    aviao.posicao.X += 5;

if (estadoTeclado.IsKeyDown(Keys.Left) && aviao.posicao.X > 0)
    aviao.posicao.X -= 5;

if (estadoTeclado.IsKeyDown(Keys.Up) && aviao.posicao.Y > 0)
    aviao.posicao.Y -= 5;

if (estadoTeclado.IsKeyDown(Keys.Down) && aviao.posicao.Y <
    Window.ClientBounds.Height - aviao.altura )
    aviao.posicao.Y += 5;
```

A próxima etapa é o disparo de tiros. Usamos aqui nossa variável de controle tempoDesdeUltimoTiro para controlarmos o tempo desde o último tiro, em seguida, verificamos se a tecla espaço, para disparar o tiro está pressionada, se o tempo desde o último tiro é maior que nossa constante de tempo de tiro, tempoTiro. Neste caso entramos no procedimento de tiro, primeiro reduzimos o tempoTiro de nossa variável de controle, e adicionamos um tiro a coleção genérica de tiros. A adição do tiro a coleção é uma chamada ao construtor de nossa classe Sprite, onde, para a posição de disparo do tiro, temos que pegar a posição do avião. Por fim, chamamos a Cue para disparar o som do tiro. Veja que no método de estamos estudando, Update, nada é desenhado, o desenho é feito no método Draw.

```
//controle do tiro, de acordo com o tempo
tempoDesdeUltimoTiro +=
```

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
gameTime.ElapsedGameTime.Milliseconds;
if (estadoTeclado.IsKeyDown(Keys.Space) &&
    tempoDesdeUltimoTiro > tempoTiro )
{
    tempoDesdeUltimoTiro -= tempoTiro;
    tiro.Add(new Sprite(Content.Load<Texture2D>(@"Imagens\Tiro"),
        new Vector2(aviao.posicao.X + aviao.largura / 2,
            aviao.posicao.Y)));
    somTiro = bancoSom.GetCue("Tiro");
    somTiro.Play();
}
```

Assim como criamos tiros, temos que atualizar a posição dos mesmos a cada frame e remover os tiros que passaram do limite da tela. Haverá outro ponto de remoção de tiros que será na detecção de colisão. O tiro do avião percorre da posição que o avião se encontra em direção a parte superior da tela, ou seja, seu valor Y deve ser decrementado. Este bloco de código percorre toda a coleção de tiros ativos e atualiza a posição. Para os tiros que já estão fora dos limites da tela, ou seja, tem sua posição Y negativa, removemos da coleção (Usamos -10 ao invés de -1 como ponto de remoção para o usuário não ver o tiro “desaparecer” da tela).

```
//remove tiros que ultrapassaram os limites da tela
//atualiza a posição dos tiros ativos
foreach (Sprite s in new List<Sprite>(tiro))
{
    s.posicao.Y -= 10;
    if (s.posicao.Y < -10)
        tiro.Remove(s);
}
```

Agora vamos gerar os inimigos. Primeiro um laço verifica quantos inimigos devem ser gerados, lembrando que temos uma constante `maximoInimigos` com a informação da quantidade máxima de inimigos ativos. Tudo a fazer, é percorrer o laço enquanto este for menor que a o máximo de inimigos que devam existir menos os inimigos existentes, valor obtido com a propriedade `Count` da coleção de inimigos. Em jogos é comum a necessidade da geração de números aleatórios, para isto criamos a função `GeraAleatorio` que estudaremos mais adiante, ela recebe um valor mínimo e máximo. O valor gerado será inferior ao máximo passado como parâmetro. `posx` é uma variável que receberá um número aleatório entre 1 e a largura da tela, desta forma os inimigos serão gerados todos em posições aleatórias. O módulo da mesma variável é utilizado para “sortearmos” o tipo de inimigo: Linear ou inteligente. Em seguida carregamos o sprite do inimigo, conforme o tipo definido, e adicionamos o inimigo a coleção genérica:

```
//gera inimigos
for (int i = 0; i < maximoInimigos - inimigo.Count; i++)
{

    //variável para posição aleatoria para o sprite
    int posx = GeraAleatorio(1, Window.ClientBounds.Width);

    //Usa mesma variável para definir aleatoriamente o tipo
    do inimigo
    TipoInimigo tipo = (TipoInimigo)(posx % 2);
```

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
string spriteInimigo;

if (tipo == TipoInimigo.Linear)
    spriteInimigo = @"Imagens\Inimigo";
else
    spriteInimigo = @"Imagens\InimigoInteligente";

//adiciona o inimigo a lista, a posição Y -100 é para o
//usuário não ver o inimigo sendo adicionado ao jogo
inimigo.Add(new Sprite(Content.Load<Texture2D>(spriteInimigo), new
    Vector2(posx, -100), tipo));
}
```

Assim como inimigos são adicionados, inimigos tem que ter sua posição atualizada e devem ser removidos se ultrapassaram os limites da janela. A lógica é semelhante à de atualização e remoção de tiros, porém, note que inimigos lineares tem simplesmente sua posição Y incrementada, enquanto que para inimigos inteligentes, além de se atualizar a posição Y com um valor maior, o código compara a posição Y do inimigo com a da avião do jogador, de forma a movimentá-lo na direção a este. Isto em tempo de jogo fará com que o Inimigo inteligente persiga o avião do jogador. Criamos assim uma forma simples e rudimentar, mas funcional, de inteligência artificial para nossos inimigos inteligentes.

```
//remove inimigos que ultrapassaram os limites da tela
//atualiza a posição dos inimigos ativos
foreach (Sprite s in new List<Sprite>(inimigo))
{
    if (s.tipoInimigo == TipoInimigo.Linear)
        s.posicao.Y += 1;
    else
    {
        s.posicao.Y += 3;
        if (s.posicao.X < aviao.posicao.X)
            s.posicao.X += 2;
        else
            s.posicao.X -= 2;
    }

    if (s.posicao.Y > Window.ClientBounds.Height)
        inimigo.Remove(s);
}
```

O que falta agora para nosso método Update é verificar colisões. Existem dois tipos de colisões possíveis: Entre os tiros do avião do jogador e do inimigo, e entre os inimigos e o avião do jogador, faremos a detecção de colisão entre estes dois tipos em blocos de código separados.

Primeiro vamos à colisão entre inimigos e tiros. Como ambos são coleções, temos que percorrer ambas. Dentro dos laços chamamos a função de colisão da classe sprite, passado o tiro como parâmetro. Se tivermos um resultado positivo, adicionamos uma explosão a coleção de explosões de acordo com a posição do inimigo que esta explodindo, tocamos o som da colisão, removemos o tiro, afinal ele não deve prosseguir para explodir outros inimigos, e

Curso de XNA Desenvolvimento de jogos – Módulo 7

adicionamos os pontos ao jogador conforme o tipo do inimigo e as constantes de pontos equivalentes.

```
//verifica colisao entre inimigos e tiros
foreach (Sprite s in new List<Sprite>(inimigo))
{
    foreach (Sprite t in new List<Sprite>(tiro))
    {
        if (s.DetectaColisao(t))
        {
            explosao.Add(new Sprite(Content.Load<Texture2D>
                (@"Imagens\Explosao"), new Vector2(s.posicao.X, s.posicao.Y),
                0));
            somColisao = bancoSom.GetCue("Explosao");
            somColisao.Play();
            inimigo.Remove(s);
            tiro.Remove(t);
            if (s.tipoInimigo == TipoInimigo.Inteligente)
                pontos += pontosInimigosInteligentes;
            else
            {
                pontos += pontosInimigosLineares;
            }
        }
    }
}
```

E quando o avião do jogador colide? Aqui temos várias coisas interessantes, como a troca do estado do jogo para PerdeuVida ou Encerrado. Primeiramente agora precisamos de apenas um laço, já que o avião do jogar é único. Quanto à colisão como avião é detectada, vários procedimentos devem ser executados: Disparamos o som da explosão, reposicionamos o avião para a posição inicial, removemos tiros, explosões e inimigos que ainda estavam ativos, e finalmente verificamos qual será o novo estado do Jogo. Se não restam mais vidas, o estado do jogo passa para encerrado, os pontos são zerados e as vidas restauradas. O jogo está pronto para recomeçar. Se ainda restam vidas, retiramos uma vida, alteramos o estado do jogo para PerdeuVida. O código break serve para sair imediatamente do bloco de código e não remover mais de uma vida do usuário.

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
//verifica colisao entre inimigos e a nave
foreach (Sprite n in new List<Sprite>(inimigo))
{
    if (aviao.DetectaColisao(n))
    {
        somColisao = bancoSom.GetCue("Explosao");
        somColisao.Play();

        aviao.posicao = new Vector2(Window.ClientBounds.Width / 2,
            Window.ClientBounds.Height / 2);

        //remove todos os inimigos que ficaram ativos
        inimigo.Clear();

        //remove todos os tiros que ficaram ativos
        tiro.Clear();

        //remove todas as explosões que ficaram ativas
        explosao.Clear();

        //Trata fim de jogo e perda de vidas
        if (quantidadeVidas == 0)
        {
            estadoJogo = EstadoJogo.Encerrado;
            quantidadeVidas = 3;
            pontos = 0;
        }
        else
        {
            --quantidadeVidas;
            estadoJogo = EstadoJogo.PerdeuVida;
            break;
        }
    }
}
}
```

Draw: PerdeuVida

No método Update criamos novos inimigos, atualizamos a posição de tudo na tela, verificamos colisões e disparamos explosões. Agora é hora de atualizar a tela para o jogador desenhando nossos sprites nas devidas posições. Este método também é estruturado através de uma estrutura switch de acordo com o estado do jogo.

Curso de XNA Desenvolvimento de jogos – Módulo 7

```
switch (estadoJogo)
{
    case EstadoJogo.PerdeuVida:
        break;
    case EstadoJogo.Encerrado:
        break;
    case EstadoJogo.EmAndamento:
        break;
    default:
        break;
}
```



Como a perda de vida leva poucos instantes não desenharemos nada na tela durante este estado.

Draw: Encerrado

```
switch (estadoJogo)
{
    case EstadoJogo.PerdeuVida:
        break;
    case EstadoJogo.Encerrado:
        break;
    case EstadoJogo.EmAndamento:
        break;
    default:
        break;
}
```



Para o estado encerrado, a única ação é desenhar a tela de splash:

```
//Desenha a tela de splash
splash.Desenhar(spriteBatch);
```

Draw: EmAndamento

```
switch (estadoJogo)
{
    case EstadoJogo.PerdeuVida:
        break;
    case EstadoJogo.Encerrado:
        break;
    case EstadoJogo.EmAndamento:
        break;
    default:
        break;
}
```



Novamente a ação maior esta no estado EmAndamento.

Curso de XNA Desenvolvimento de jogos – Módulo 7

Tudo a fazer agora é desenhar nossos sprites: Primeiro o avião, os inimigos, os tiros, as explosões, as vidas e os pontos! No caso de listas, é preciso percorrer as mesmas para que o XNA desenha uma a uma. Veja que as explosões devem ser desenhadas em frames, caso tenha chegado ao último frame, ela deve ser removida da lista. Já as vidas são desenhadas de acordo com a quantidade de vidas que restam ao usuário.

```
//Desenha o avião do jogador
aviao.Desenhar(spriteBatch);

// Desenha todos os inimigos existentes
foreach (Sprite s in new List<Sprite>(inimigo))
{
    s.Desenhar(spriteBatch);
}

// Desenha tiros
foreach (Sprite s in new List<Sprite>(tiro))
{
    s.Desenhar(spriteBatch);
}

//Desenha as explosões
foreach (Sprite s in new List<Sprite>(explosao))
{
    if (s.frameAtual <= 6)
    {
        s.Desenhar(spriteBatch, new Rectangle(s.frameAtual *
        s.largura / 7, 0, s.largura / 7,
        s.altura));
        ++s.frameAtual;
    }
    else
    {
        explosao.Remove(s);
    }
}

//Desenha as vidas
vidas.Desenhar(spriteBatch, new Rectangle(0, 0,
quantidadeVidas * vidas.largura / 3, vidas.altura));

//Desenha os pontos
spriteBatch.DrawString(fonte, pontos.ToString(), new
Vector2(0, 20), Color.White);
```

Para finalizar, nossa função de geração de números aleatórios, já conhecida, que recebe um valor mínimo e máximo para a geração. Para o método random devemos fornecer uma “semente”, onde passamos a um tipo DateTime.

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

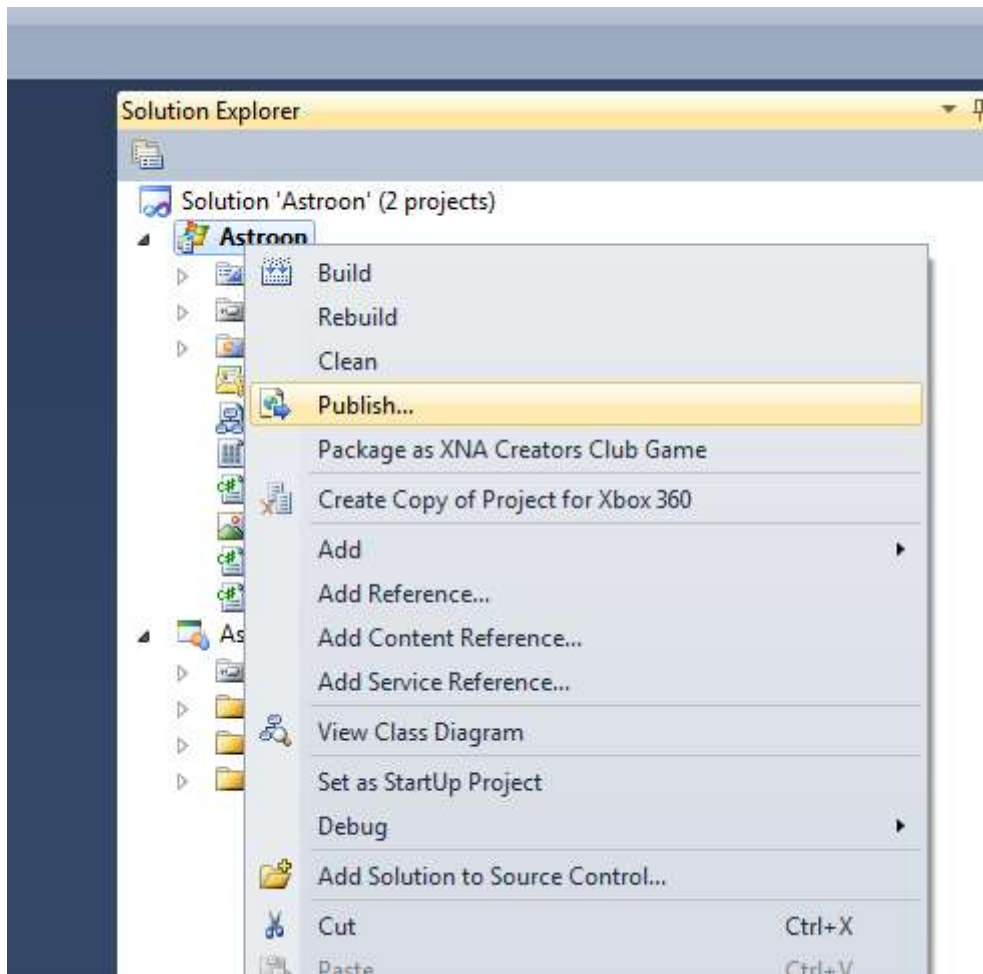

Curso de XNA Desenvolvimento de jogos – Módulo 7

}

Pronto, você já pode rodar o seu jogo!

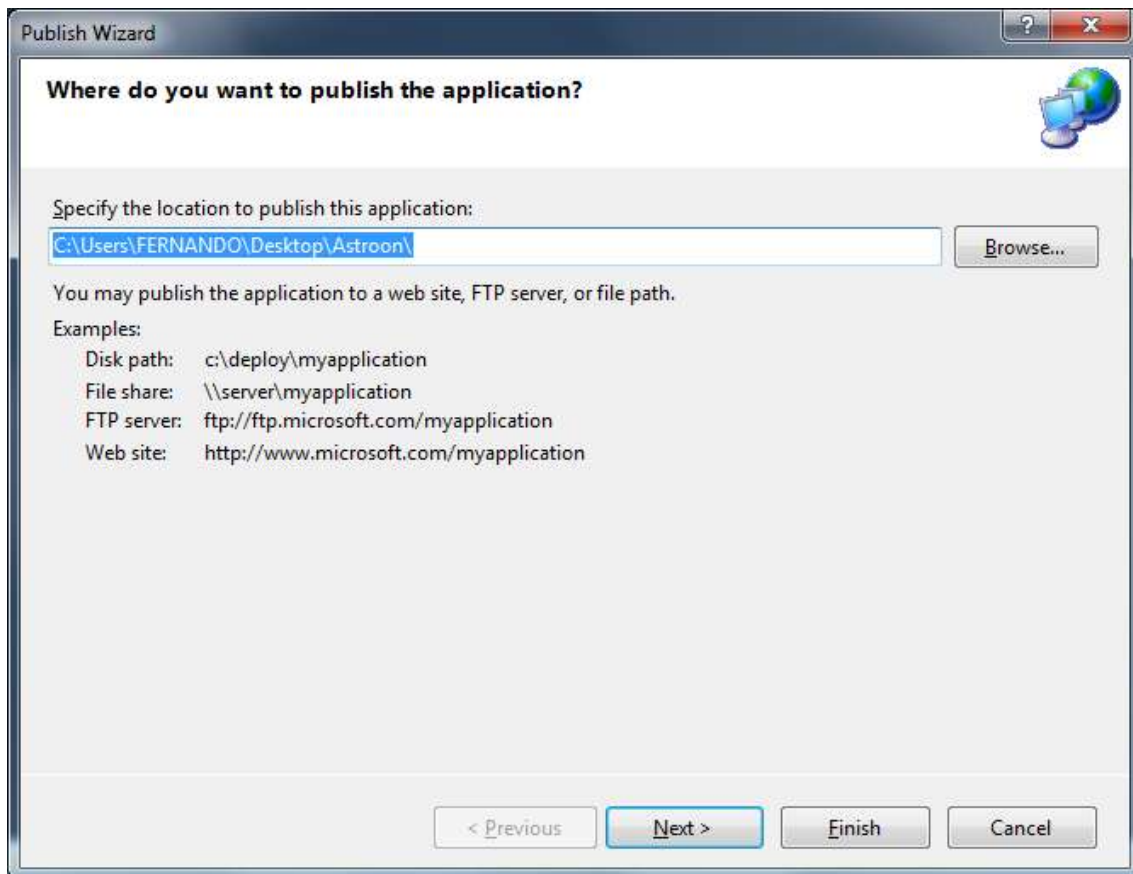
Distribuindo seu Jogo

Depois do jogo pronto você precisará distribuí-lo. Em jogos para PC, o usuário deverá ter instalado o .NET Framework 4.0. Não basta enviar o executável gerado pelo XNA, você deverá executar um pequeno procedimento para gerar um instalador. Clique com o botão direito no projeto do seu jogo e selecione Publish:



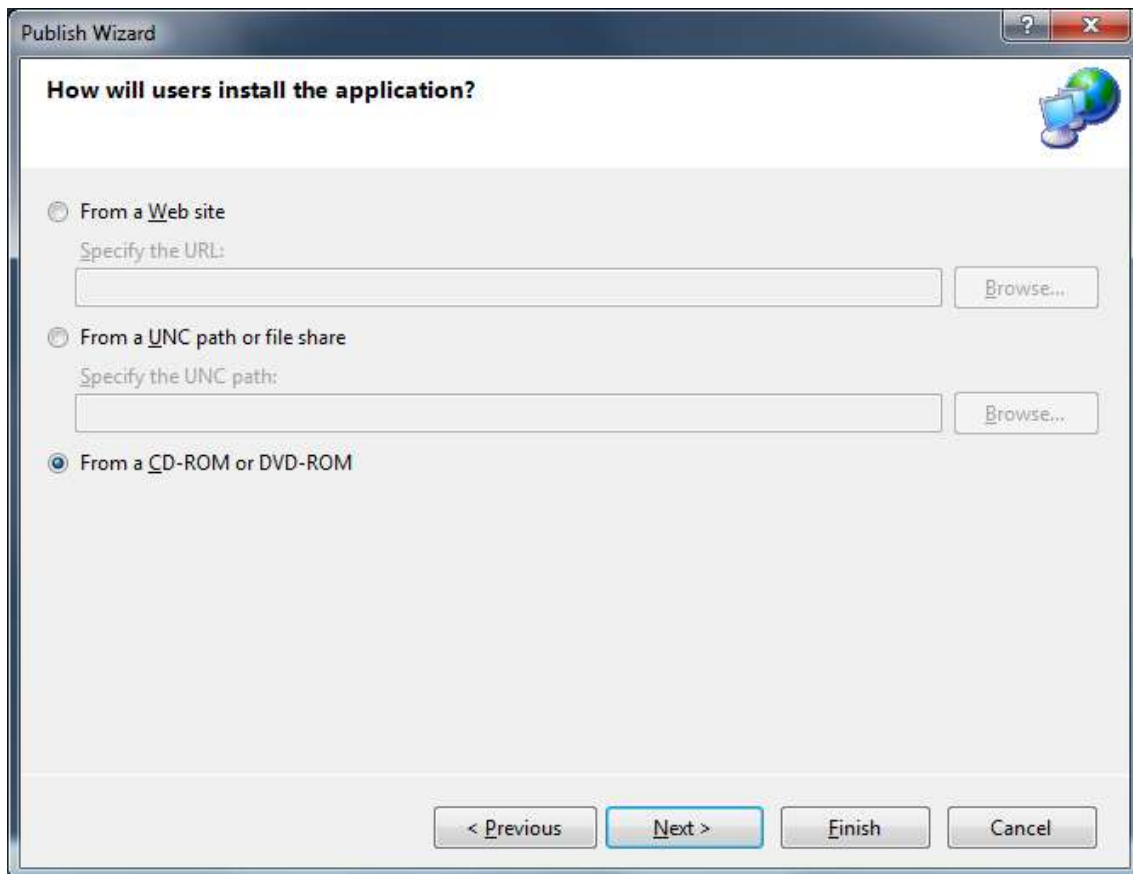
Defina o local onde os arquivos deverão ser gerados:

Curso de XNA Desenvolvimento de jogos – Módulo 7



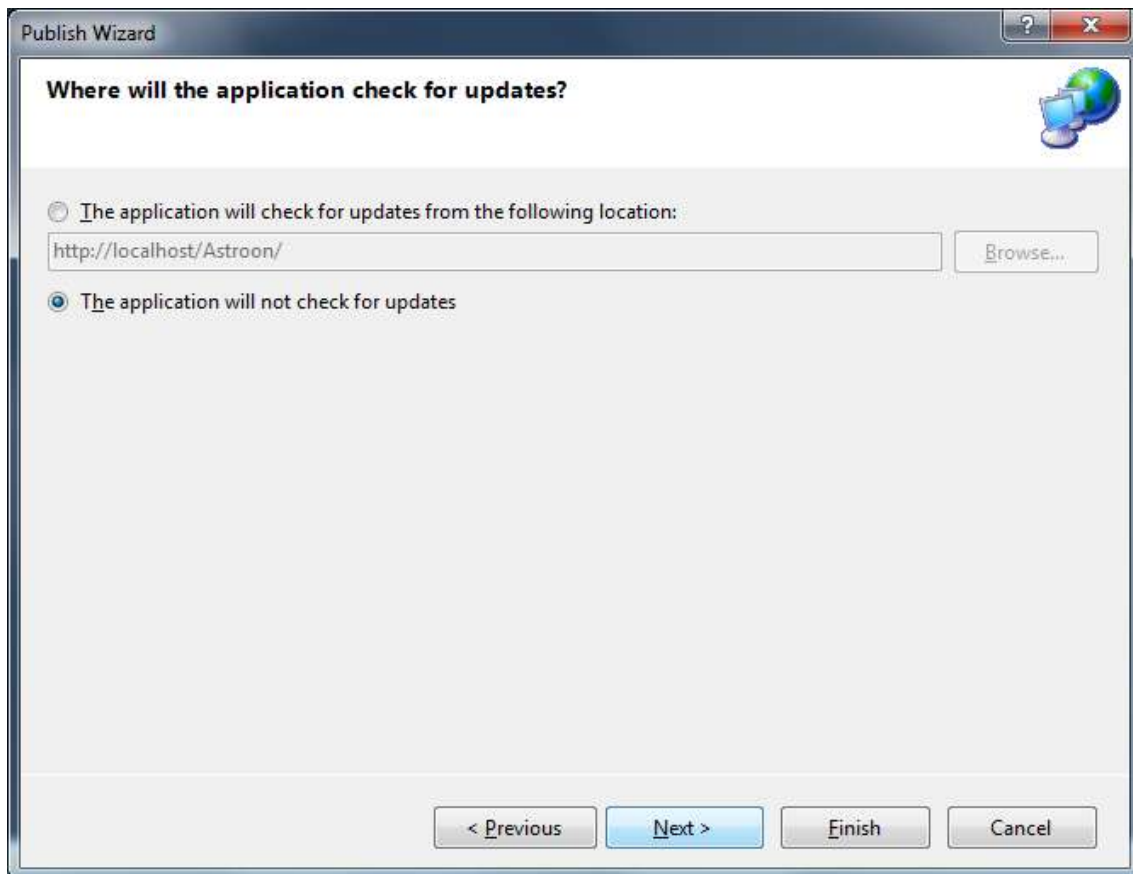
Defina como a aplicação será acessada: Escolhemos a opção CD-ROM ou DVD-ROM:

Curso de XNA Desenvolvimento de jogos – Módulo 7



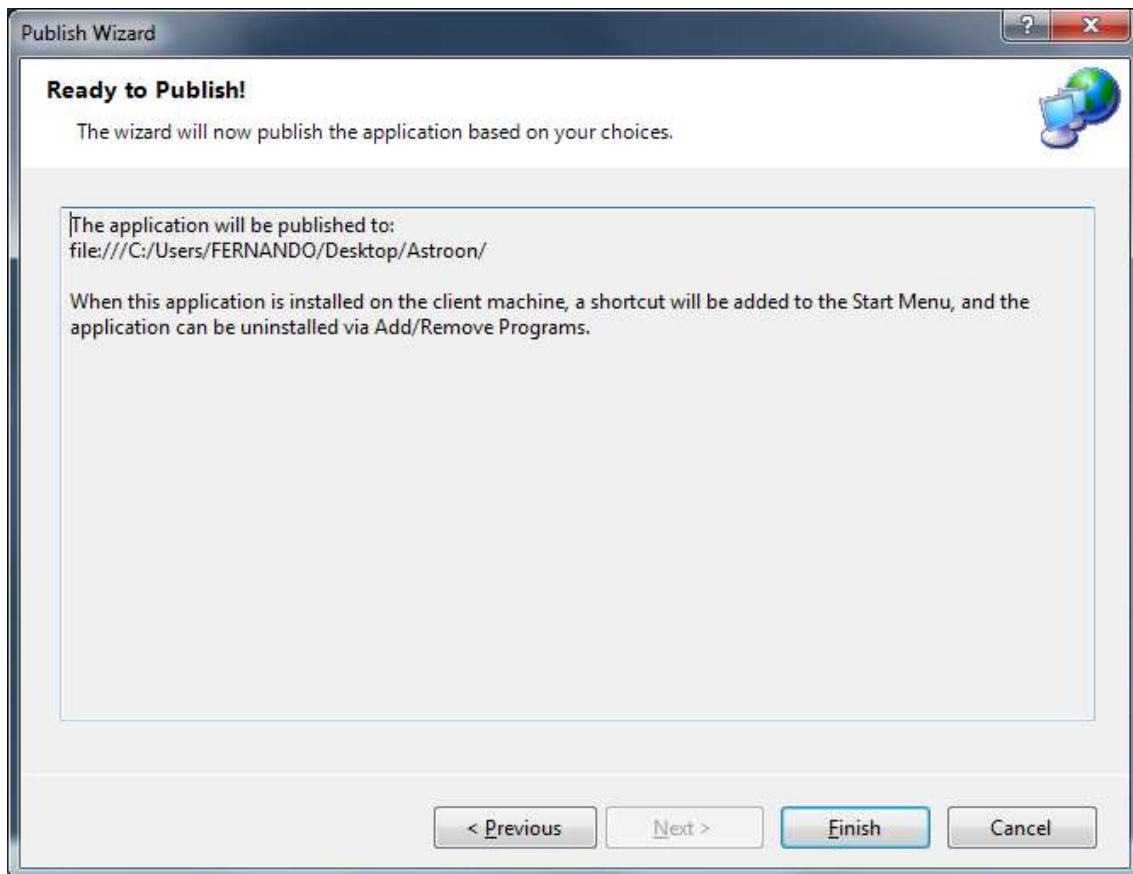
Defina se a aplicação irá verificar atualizações. Marque não.

Curso de XNA Desenvolvimento de jogos – Módulo 7

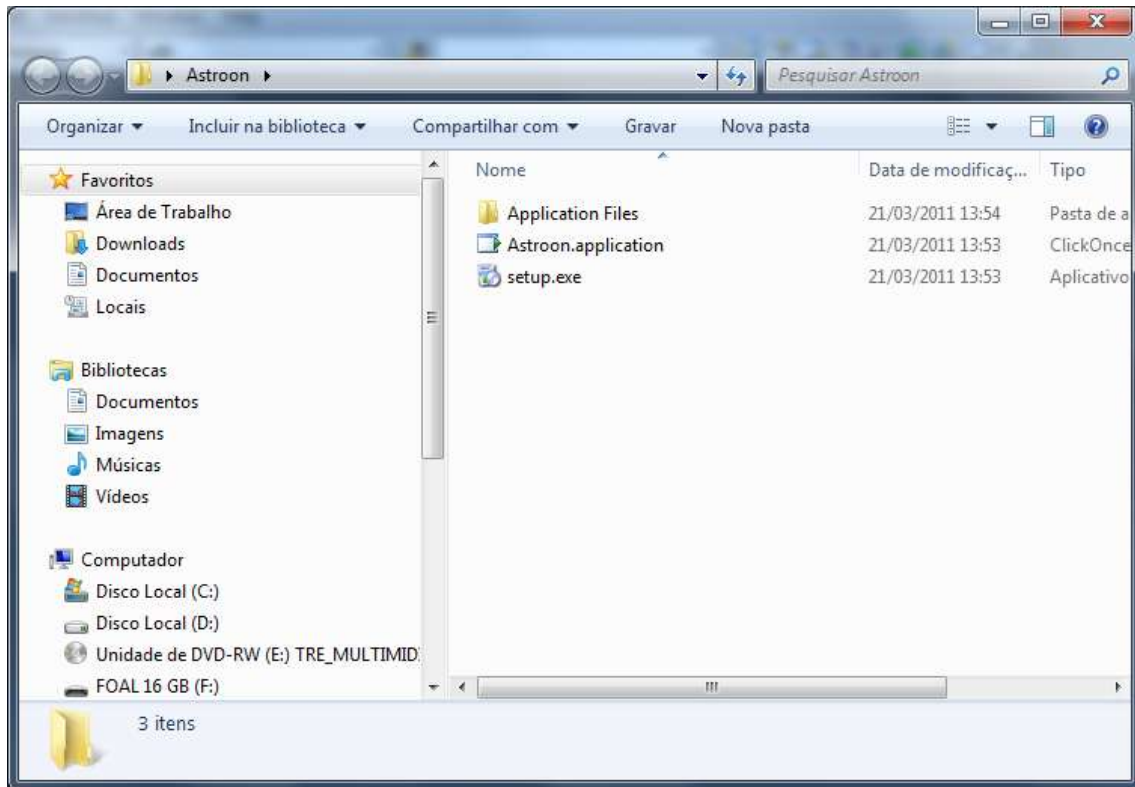


O XNA apresenta um resumo da publicação:

Curso de XNA Desenvolvimento de jogos – Módulo 7



Os arquivos são gerados na pasta indicada. Para o usuário instalar o jogo, basta clicar sobre setup.exe. Lembre-se que este instalador não instala o .NET Framework 4.0, o usuário deverá instalá-lo a parte, ou você deve criar outro tipo de instalador, como um projeto de Setup e Deployment do próprio Visual Studio.



Indo Além

Você pode ir muito além no projeto de Jogo do Astroon: Você pode adicionar fases, aumentando a dificuldade entre as fases, adicionar novos tipos de inimigos, fazer com que os inimigos revidem os seus ataques e atirem em você, você pode criar inimigos “chefes” em determinadas fases, pode construir um cenário de fundo (ilhas e navios passando) enquanto o jogo roda etc. Ou seja, o limite é sua imaginação. Procuramos manter o projeto funcional e ao mesmo tempo simples, para ser didático e facilitar o aprendizado.

De qualquer forma, montamos também uma nova versão do Astroon com as seguintes características:

- **Controle de fases, com aumento de dificuldade**
- **Novos tipos de inimigos, adicionados no decorrer das fases**
- **Inimigos contra-atacam (atiram)**
- **Inimigo chefe a cada 5 fases**
- **Cenário de Fundo**
- **Algoritmo de colisão mais preciso**

Curso de XNA Desenvolvimento de jogos – Módulo 7

Como não faz parte do escopo deste curso, esta versão mais completa não foi publicada. Porém, se você quiser conhecê-la e estudá-la por sua própria conta e risco, poste uma mensagem no fórum do curso que colocaremos a disposição do aluno esta versão mais incrementada do nosso jogo.

Conclusão

Parabéns, você concluiu o curso de XNA 4.0. O próximo passo é fazer a prova on-line e obter seu certificado. Se você estudou todo o conteúdo do curso e codificou e entendeu todos os exemplos, não terá qualquer dificuldade.

Até a próxima e sucesso com seus jogos!