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

Conteudista: André Luiz Brazil

Aula 5: INICIANDO O PROJETO DE UM JOGO NO VISUAL C#

META

Criar e testar o código de um projeto de jogo produzido através das ferramentas Visual C# e XNA Game Studio.

OBJETIVOS

Ao final da aula, você deve ser capaz de:

1. Criar um novo projeto de jogo com as ferramentas Visual C# e XNA Game Studio;
2. Entender o funcionamento dos principais métodos existentes em um projeto de jogo criado a partir das ferramentas Visual C# e XNA Game Studio;
3. Incorporar a classe Espaçonave, vista na aula 4, dentro do código do projeto de 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. Conhecer os conceitos de classe e objeto, abordados na aula 4.

Introdução

Na aula anterior aprendemos como instalar as ferramentas Visual C# e XNA Game Studio, necessárias para a criação de um projeto de jogo. Agora que você possui um plano de jogo e as ferramentas disponíveis no computador, você já pode iniciar o desenvolvimento do seu projeto de jogo.

Como faremos então para criar um novo projeto de jogo dentro do Visual C#?

Precisamos entrar no ambiente Visual C# e iniciar um novo projeto. Para tal:

1) A partir do menu Iniciar do Windows, chame a ferramenta Visual C#. Em seguida, aparecerá a tela inicial do Visual C#, contendo:

- Uma lista de projetos recentemente acessados pela ferramenta no canto esquerdo, dentro da seção "*Recent Projects*";
- Ainda no canto esquerdo aparecem alguns tutoriais de desenvolvimento, agrupados na seção "Get Started";
- Por último, aparecem as seções "Visual C# Headlines" (à esquerda) e "Visual C# Developer News (ao centro)", contendo informativos e notícias relacionadas à ferramenta Visual C#.

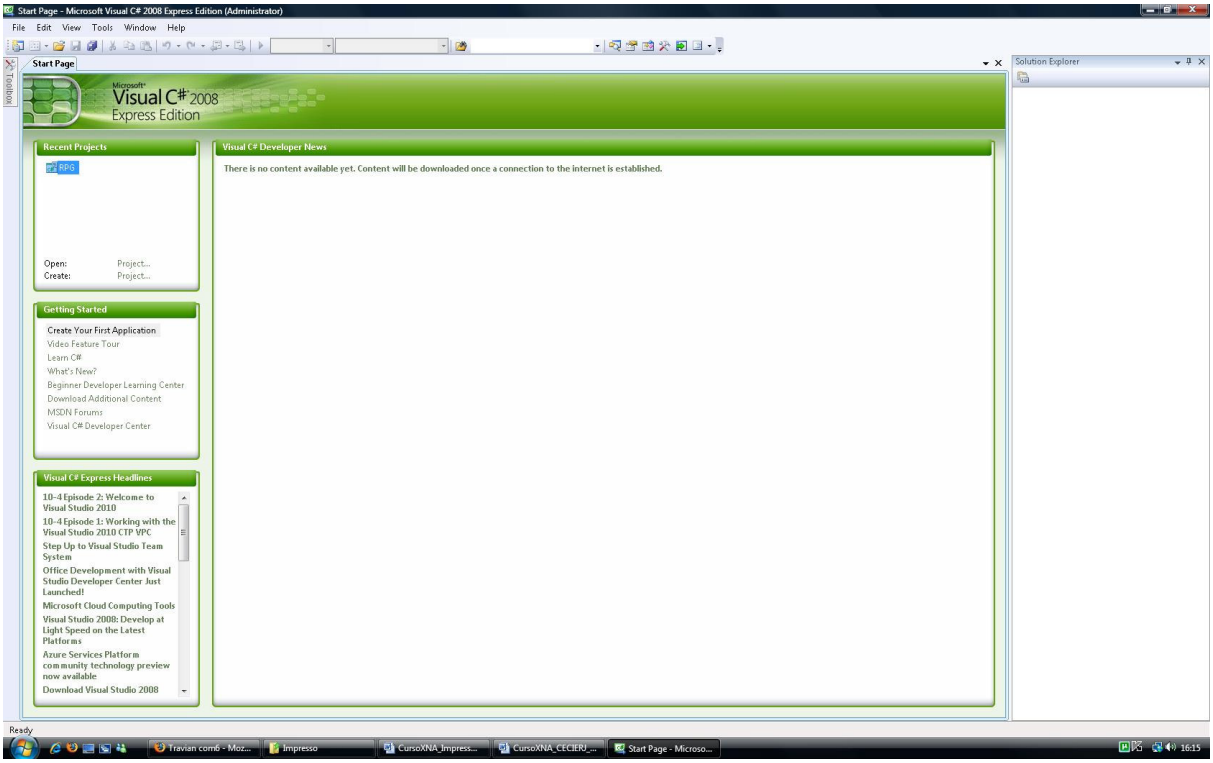

Figura 5.1 - Tela inicial do Visual C#

Figura 5.1

Fonte: Visual C#

Legenda: Tela inicial da ferramenta Visual C#

2) Selecione a opção “File” no menu superior e, em seguida, a opção “New project”, para criar um novo projeto.

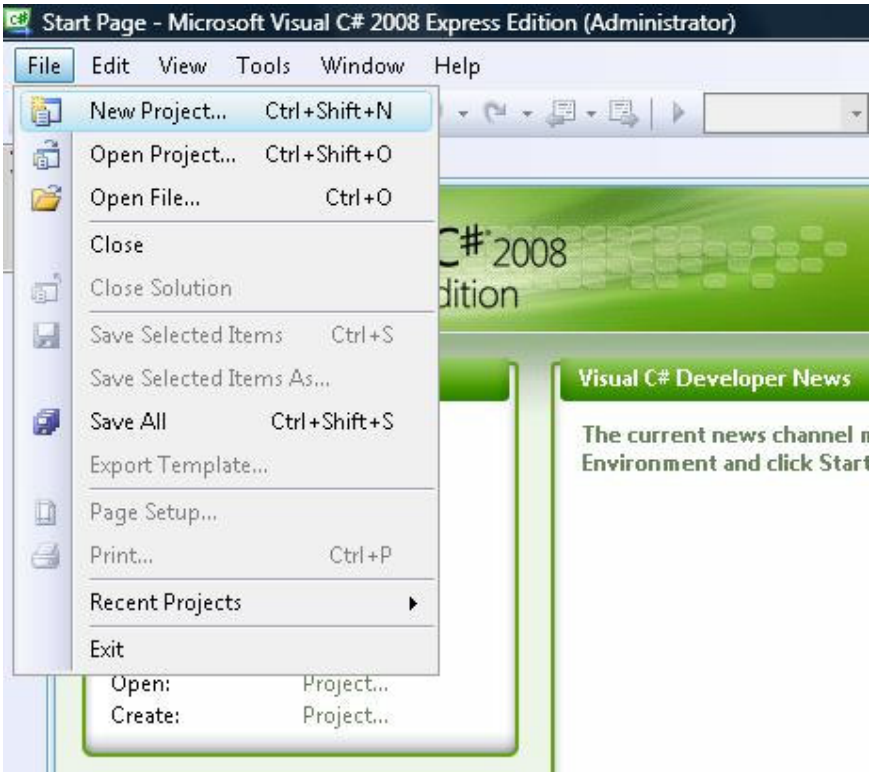

Figura 5.2 - Opção File/New Project, para criar um novo projeto

Figura 5.2
Fonte: Visual C#
Legenda: Opção File/New Project, para criar um novo projeto

3) Você verá uma tela contendo os tipos de projetos disponíveis para o Visual C#. Selecione no lado esquerdo a opção “XNA Game Studio” para listar ao lado direito os tipos de projeto correspondentes ao XNA. Por fim, selecione do lado direito o tipo de projeto “Windows Game”, ou seja, jogo para Windows. Na parte de baixo da janela aparecem os seguintes campos:

- Name - Coloque aqui o nome do seu projeto de jogo;
- Location – Indica a pasta onde será salvo o projeto. Preencha com “C:\ProjetosXNA”;
- Solution Name – Significa o nome da solução. Para projetos grandes, pode existir mais de uma solução dentro de um mesmo projeto. Para o nosso caso, deixe preenchido com o mesmo nome do projeto.

Agora aperte o botão “OK” para confirmar a criação do seu projeto de jogo!

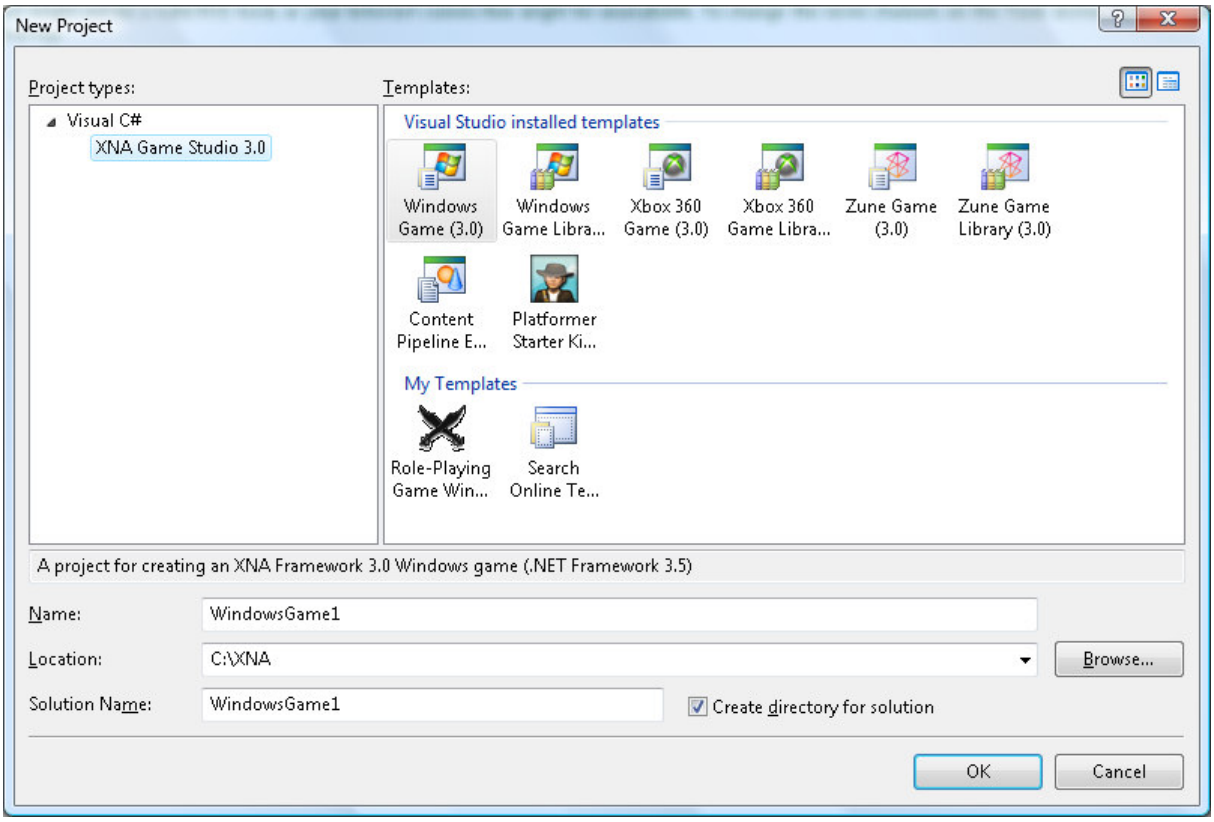

Figura 5.3 – Criação de um novo projeto de jogo para Windows em XNA

Figura 5.3

Fonte: Visual C#

Legenda: Criação de um novo projeto de jogo para Windows em XNA

Atividade 1 – atende ao objetivo 1

Siga os três passos acima e crie um projeto de jogo chamado “JogoEspacial”, localizado na pasta “C:\ProjetosXNA”.

Fim da Atividade 1

Entendendo a estrutura de um projeto de jogo

Após ter criado o seu projeto de jogo, você verá uma janela contendo um código um pouco estranho. É o código de programação do projeto JogoEspacial, que você acabou de criar, escrito na linguagem C#. E agora, o que significa cada parte desse código?

Logo no início, podemos observar uma série de instruções contendo a palavra “**using**” escrita na cor azul. Veja só:

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
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;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
```

Traduzindo para o português, a palavra using significa usando. Isso mesmo! Chamamos o comando "**using**" quando precisamos usar, ou seja, incluir dentro do nosso código outras bibliotecas de comandos.

Verbete

Biblioteca de comandos ou funções é um arquivo que contém várias instruções agrupadas, prontas para serem utilizadas dentro de outros projetos.

Fim do Verbete

Por exemplo, ao escrevermos:

```
using Microsoft.Xna.Framework.Audio;
```

Imagine como se você estivesse dentro de uma biblioteca chamada Microsoft e dissesse para o atendente Visual C#: "Por favor, gostaria de obter alguns livros de XNA falando sobre os sons". O atendente então lhe diria: "Siga para a sessão XNA e dentro dela procure uma prateleira rotulada Framework. Nela você encontrará os livros de áudio".

Figura 5.4

Fonte: http://www.ccm.edu/library/OLD%20WEB/library_image.jpg (Jefferson, favor redesenhar)

Desta forma, ao chamar a instrução acima, estamos dizendo para o Visual C# que queremos ter disponíveis todos os comandos da biblioteca Microsoft dentro das seções XNA e Framework que estejam relacionados ao áudio, ou seja, comandos prontos para lidar com os sons do nosso jogo.

Verificando os demais comandos "**using**", podemos perceber que estamos tornando disponíveis para o nosso jogo diversas instruções de XNA relacionadas ao áudio, conteúdo (content), gráfico (graphics), entrada (input), rede (net), armazenamento (storage) e outros, que certamente serão necessárias na produção do jogo.

Repare também que apenas as palavras "**using**" aparecem na cor azul. Isto significa que o Visual C# reconheceu essa palavra e sinalizou para você com a cor azul, indicando que "**using**" é uma palavra reservada.

Verbete

Palavra reservada é uma palavra com significado especial para o Visual C#. Palavras desse tipo aparecem na cor azul dentro do código do projeto e não podem ser utilizadas para dar nome aos objetos criados dentro do projeto.

Fim do Verbete

Agora vamos ver o que significa uma outra palavra reservada, chamada "**namespace**":

```
namespace JogoEspacial
{
```

Traduzindo para o português, "namespace" significa "nome de espaço". Ao utilizarmos a instrução "**namespace**" JogoEspacial, estamos mostrando para o Visual C# que criamos uma área chamada "JogoEspacial", onde colocaremos instruções do nosso jogo. Normalmente o "**namespace**" possui o mesmo nome do projeto. Aqui dentro ficará todo o código do nosso jogo.

Repare que na linha abaixo, após o comando "**namespace**", existe a chave "{". As chaves "{" e "}" indicam o início e o fim desse espaço de desenvolvimento para o Visual C#.

Veja o trecho de código abaixo. Lembra-se do conceito de classe que aprendemos durante a aula 4?

```
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
```

Aqui encontramos uma nova classe, chamada Game1. Repare que, ao lado do nome da classe, aparece o sinal de dois pontos (**:**), e depois aparece também o nome de uma outra classe. Isto quer dizer que a classe Game1 é filha da classe Microsoft.Xna.Framework.Game. Desta forma, a classe Game1 irá possuir todos os atributos e métodos que a classe Microsoft.Xna.Framework.Game, a classe pai, possui. A esta propriedade, damos o nome de herança.

Verbete

Herança é uma propriedade que indica que todos os métodos e atributos de uma classe pai pertencerão automaticamente também à classe filha.

Fim do Verbete

Repare agora no trecho seguinte:

```
GraphicsDeviceManager graphics;
SpriteBatch spriteBatch;
```

Estes são os atributos novos da classe Game1. São eles:

- **graphics**, do tipo GraphicDeviceManager – é o atributo que representa o nosso dispositivo de vídeo, onde iremos exibir as imagens do jogo;
- **spriteBatch**, do tipo SpriteBatch – foi criado para armazenar um conjunto de sprites, ou seja, imagens para serem utilizadas no nosso jogo.

Veja agora o método de criação da classe Game1:

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    content = new ContentManager(Services);
}
```

Dentro do método de criação da classe Game 1, estamos inicializando os dois atributos novos da classe. Ambos os atributos **graphics** (vídeo) e **content** (conteúdo) são atributos de tipo complexo e precisam da instrução **new** para serem efetivamente criados. Dentre os tipos simples, podemos listar aqueles que vimos na aula 4, que são o **int** (número inteiro) e o **string** (conjunto de caracteres). Os atributos de tipo simples não precisam da instrução **new** para serem inicializados.

Observe agora o método Initialize:

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

    base.Initialize();
}
```

Aha! Aqui está a grande diferença entre a estrutura do XNA e a das outras ferramentas de produção de jogos. A organização. O método Initialize e os outros abaixo foram criados para ajudar você, programador, a estruturar melhor o seu jogo.

Este método foi criado para você colocar valores iniciais para os objetos do seu jogo. Você poderia, por exemplo, colocar aqui dentro energia_inicial = 50, indicando que todas as espaçonaves teriam 50 pontos de energia ao iniciar o jogo.

Existe ainda a palavra **override**, que indica que esse método **Initialize** está substituindo um outro método com o mesmo nome **Initialize**. Como assim? É que quando herdamos os atributos e métodos da classe **Microsoft.XNA.Network.Game**, ela já possuía esse método Initialize também. Portanto, estamos substituindo o método Initialize da classe **Microsoft.XNA.Network.Game** pelo nosso próprio método Initialize.

Dentro do método Initialize, aparece também, a instrução **base.Initialize()**. Esta instrução serve para chamar o método Initialize da classe pai, a **Microsoft.XNA.Network.Game**. Ela foi colocada aqui para não perdermos tudo o que foi codificado no método **Initialize** original, que foi herdado da classe **Microsoft.XNA.Network.Game**.

Abaixo os métodos **LoadContent** e **UnloadContent**:

```
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
}

protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}
```

Estes métodos, foram criados para você colocar dentro deles todos os objetos gráficos a serem carregados (load) ou descarregados (unload) dentro do seu jogo, incluindo imagens, texturas, modelos de personagem, etc. Veja também que ele inicializa o conjunto de sprites **spriteBatch**, deixando-o pronto para que você possa carregar suas imagens dentro dele.

Vejamos agora o método **Update**:

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

    // TODO: Add your update logic here

    base.Update(gameTime);
}
```

Este é um dos métodos mais importantes do seu jogo. Ele é responsável pela programação de toda a mecânica de funcionamento do jogo. É aqui que você verifica se o tiro do jogador acertou a nave inimiga e também se ela foi destruída, checando os pontos de vida dela, por exemplo. Aqui dentro você coloca as regras do jogo.

Também é utilizado para definir ações a serem realizadas quando o usuário apertar alguma tecla especial, como por exemplo, a tecla "Esc", para sair do jogo. É exatamente esta verificação que o XNA já coloca para você dentro do código, através da instrução:

```
if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed)
    this.Exit();
```

Isto quer dizer que se (**if**) a tecla "Esc" (**Buttons.Back**) for apertada (**ButtonState.Pressed**), o jogo se encerra (**this.Exit**).

Repare que este método recebe também um parâmetro de entrada que é o **gameTime**, ou seja, o tempo do jogo, que irá ser utilizado para controlar, entre outras coisas, de quanto em quanto tempo a tela do jogo será redesenhada, por exemplo.

Por último, o método **Draw**:

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

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
```

Este método será chamado toda vez que a tela do jogo precisar ser redesenhada, seja porque o tempo passou e a tela precisa ser atualizada com as naves e os tiros desenhados em uma nova posição ou então para exibir a famosa mensagem "Game Over", quando o jogo acabar.

Qualquer instrução para atualizar graficamente o cenário ou os personagens do jogo deve ser incluída dentro deste método.

Repare que o XNA já coloca um comando dentro desse método que é o **GraphicsDevice.Clear**. Este comando limpa a tela preenchendo o fundo com a cor que você quiser, no caso foi definida a cor azul (**Color.CornflowerBlue**). Para um jogo espacial, seria mais apropriada a cor preta, não acham?

Atividade 2 – atende ao objetivo 2

Altere o código do seu jogo e mude a cor de fundo da tela para uma outra cor qualquer, de acordo com o seu plano de jogo. Escreva abaixo como ficou o seu método Draw, após as alterações.

_____

_____

_____

_____

_____

_____

Fim da Atividade 2

Incorporando a classe Espaçonave ao código do jogo

Agora que você já estudou um pouco o código do projeto de jogo, vamos criar, dentro do jogo, a espaçonave do jogador.

Vamos rever o código da classe Espaçonave:

```
public class Espaconave
{
    private string _nome;
    private int _energia;
    private int _velocidade;
    private int _potencia_tiro;
    private int _velocidade_tiro;
    private int _escudo;

    public Espaconave()
    {
        _energia = 50;
        _velocidade = 5;
        _velocidade_tiro = 5;
    }

    public void Mover(int direcao)
    {
        // Move a espaçonave na direção especificada
    }

    public void Atirar()
    {
        // Cria um tiro
    }
}
```

Para você incluir este código dentro do seu projeto de jogo, basta colocá-lo logo abaixo da classe Game1, após a chave "}", que indica o fim da classe.

Contudo, este código apenas define a classe de objetos Espaçonave. O que estamos querendo, na realidade, é criar a nave do jogador dentro do jogo. Para tal, precisaremos:

1) Criar um objeto da classe Espaconave dentro do jogo, que vai ser a nave do jogador. Vamos dar para este objeto o nome NaveJogador. Podemos fazer isso no início

do código do jogo, onde já estão definidos todos os atributos principais do jogo (classe Game1). Veja só como ficou:

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

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

2) Inicializar o objeto NaveJogador dentro do jogo, ou seja, dar origem, efetivamente, à nave. Podemos fazer isso dentro do método de inicialização do jogo, o **Initialize**:

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

    base.Initialize();
}
```

3) Configurar os atributos iniciais da nave do jogador, ou seja, colocar um nome para a nave, estabelecer a força do tiro, a velocidade, etc. Vamos chamar a nave do jogador de “Falcão Justiceiro”. Um bom local para realizar esta tarefa é dentro do método de inicialização do jogo, o **Initialize**:

```
protected override void Initialize()
{
    // Criando efetivamente a espaçonave do jogador no jogo
    NaveJogador = new Espaconave();

    // Colocando um nome para a nave do jogador
    NaveJogador._nome = "Falcão Justiceiro";

    base.Initialize();
}
```

Repare que ao realizar o passo 3, você verá que o Visual C# sublinhou a palavra **_nome** com a cor **vermelha** no código. Veja só:

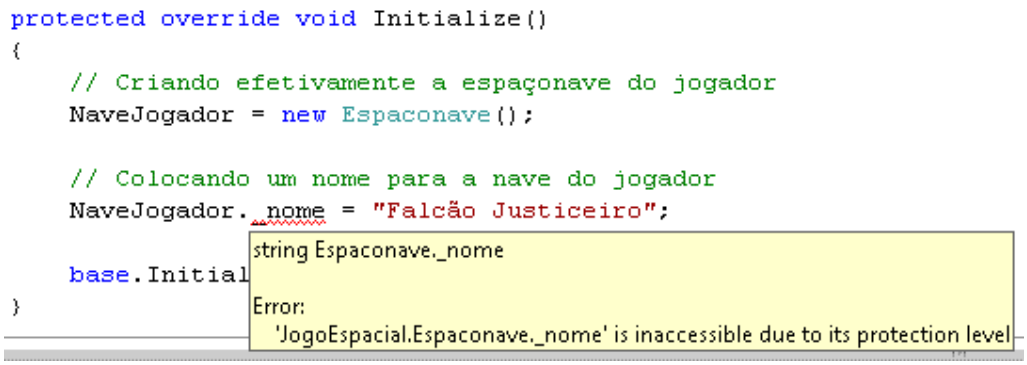

Figura 5.5
Fonte: Visual C#

Se você colocar o mouse em cima da palavra nome, verá que aparece dentro de uma caixa amarela um aviso, indicando que o "atributo **_nome** está inacessível devido ao nível de proteção dele".

O Visual C# é muito esperto e já está tentando lhe avisar de antemão que isso que você tentou fazer não vai dar certo. Porquê? Verifique na definição da classe Espaçonave como foi criado o atributo _nome:

```
private string _nome;
```

Exatamente, o atributo **_nome** foi criado como privado. Desta forma, apenas a própria classe Espaçonave conseguirá alterar o seu valor. Qual será a solução então?

Vamos criar dois novos métodos dentro da classe Espaçonave. Um para atualizar o nome da espaçonave e outro para nos trazer esse nome quando precisarmos. Veja abaixo como fazer isso:

```
// Método de atualização do atributo _nome
public void Nome(string p_nome)
{
    _nome = p_nome;
}

// Método de acesso ao valor do atributo _nome
public string Nome()
{
    return _nome;
}
```

Vamos lá! Coloque esse código dentro do seu projeto de jogo, abaixo dos outros métodos que estão dentro da classe Espaçonave.

Repare que no método de acesso ao atributo **_nome** existe o comando **return**. Este comando serve para retornar o valor do atributo **_nome**.

Perceba também que os dois métodos possuem o mesmo nome e ainda assim o Visual C# não reclamou disto. Isto ocorre porque apesar de ambos possuírem o mesmo nome, o método de atualização está recebendo um parâmetro de entrada, o **p_nome**, do tipo **string**. Já o método de acesso, não recebe nenhum valor, pelo contrário, retorna o valor **_nome**. A esta propriedade damos o nome de sobrecarga de métodos.

Verbete

Sobrecarga de métodos é uma propriedade do Visual C# que permite que vários métodos possuam o mesmo nome, desde que tenham uma forma diferente de serem chamados.

Fim do Verbete

Agora que temos o método de atualização do nome da espaçonave, vamos corrigir o nosso código do jogo para chamá-lo de forma correta:

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

    // Colocando um nome para a nave do jogador
    NaveJogador.Nome("Falcão Justiceiro");

    base.Initialize();
}
```

Atividade 3 – atende ao objetivo 3

Agora que você já aprendeu a criar os métodos de atualização e acesso para o atributo **_nome**, faça o mesmo para os demais atributos da classe Espaçonave.

Fim da Atividade 3

Que tal tentarmos verificar se está tudo certo? Para testar o código do seu projeto de jogo, selecione a opção **Build\Build Solution**, conforme a tela abaixo, ou então aperte a tecla **F6**:

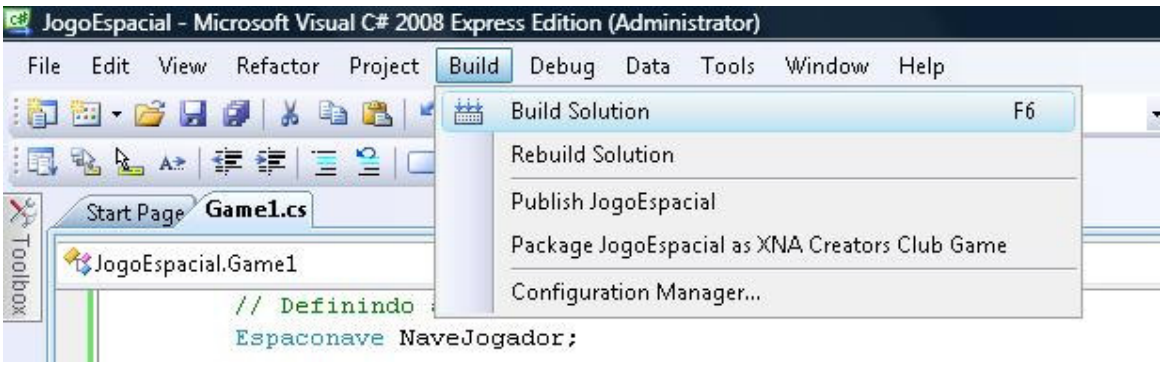

Figura 5.6

Fonte Visual C#

Caso tudo ocorra sem problemas, será exibida a mensagem "**Build succeded**" na barra de status localizada no canto inferior esquerdo da tela.

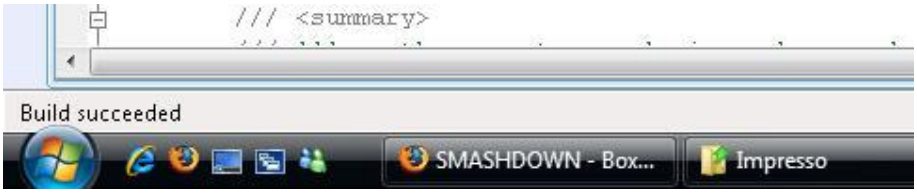

Figura 5.7

Fonte Visual C#

Caso contrário, aparecerá uma listagem dos erros encontrados, bastando que você clique duas vezes sobre os erros listados para ir até o local onde está ocorrendo o problema. Os erros impedem o código de ser executado.

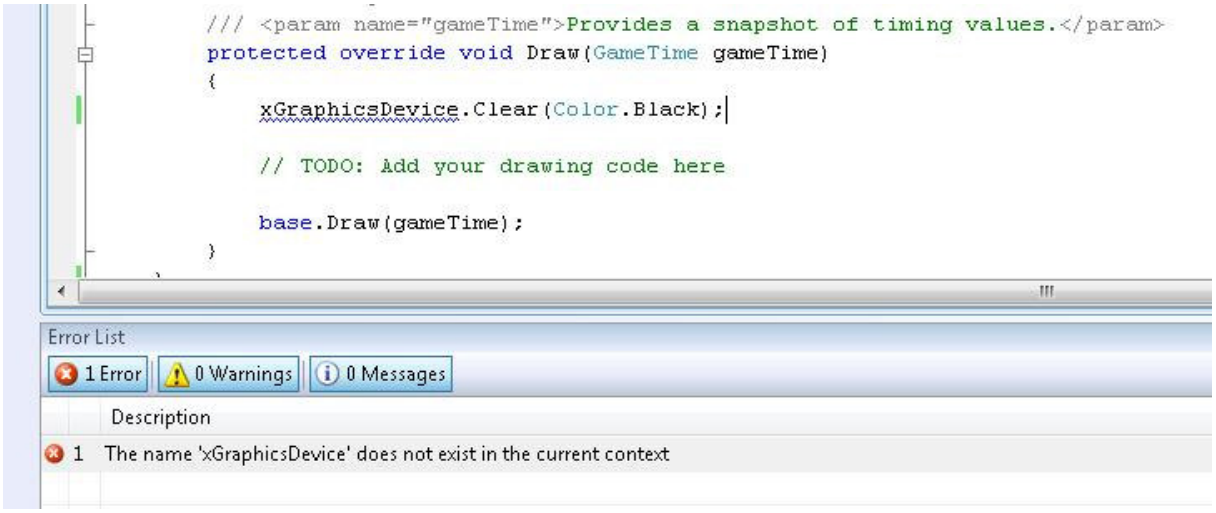

Figura 5.8

Fonte Visual C#

Podem aparecer alguns avisos (warnings), indicando, por exemplo, que algum atributo foi definido, mas não teve o seu valor alterado ou utilizado no jogo, ou outro aviso qualquer, o que geralmente não impede o código de ser executado.

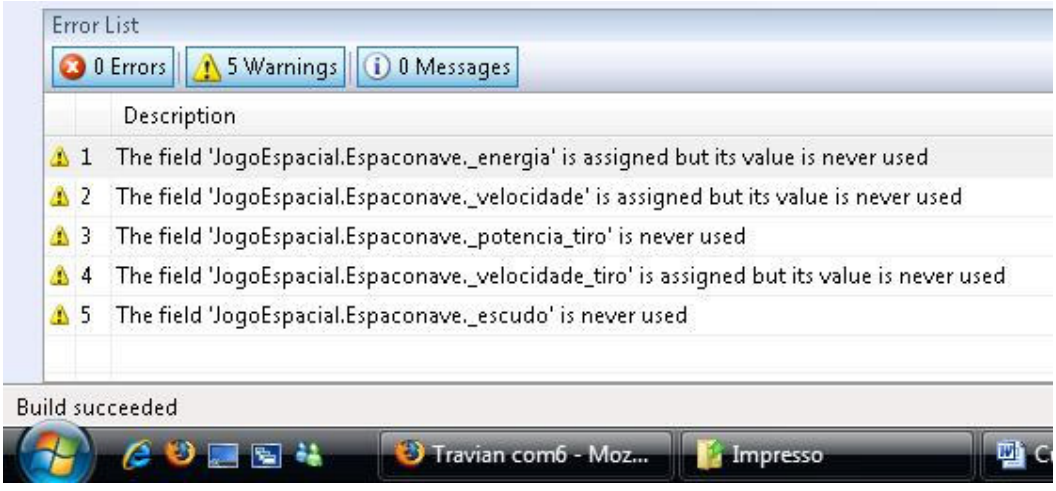

Figura 5.9

Atividade 4 – atende ao objetivo 3

Vamos testar o código do seu projeto de jogo seguindo as instruções acima? Caso o Visual C# exiba algum erro , escreva-o nas linhas abaixo e tente solucioná-lo revendo os exemplos que foram dados nas aulas anteriores e procurando ajuda no fórum. Boa sorte!

_____

_____

_____

_____

_____

_____

Fim da Atividade 4

CAIXA DE FÓRUM INFORMAÇÕES SOBRE FÓRUM

Figura 3.7
Fonte: http://www.stockxpert.com/browse_image/view/28331341/?ref=sxc_hu (Jefferson- favor redesenhar)

Você teve alguma dificuldade para codificar o seu primeiro projeto de jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os seus amigos.

FIM DE CAIXA DE FÓRUM

CAIXA DE ATIVIDADE INFORMAÇÕES SOBRE ATIVIDADE ON-LINE

Figura 3.8
Fonte: http://www.sxc.hu/photo/1000794 (Jefferson : favor redesenhar)

Agora que você já está com o seu código de projeto do jogo pronto, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

FIM CAIXA DE ATIVIDADE

Resumo

Para você criar um novo projeto de jogo no Visual C#, selecione a opção **File/New Project** no menu superior. Em seguida, aparecerá uma janela com os tipos de projetos que você pode criar. Selecione do lado esquerdo a opção **XNA Game Studio** e do lado direito a opção **Windows Game**. Em seguida, preencha os campos **Name, location e Solution Name** com os valores: "JogoEspacial", "C:\ProjetosXNA" e "JogoEspacial". Depois aperte o botão "**OK**" para criar o seu novo projeto de jogo.

Dentro do código do jogo você verá que existem várias palavras reservadas, que aparecem na cor azul, tais como **using**, **namespace**, **public** e **class**, por exemplo. Estas palavras são especiais para o Visual C# e não podem ser utilizadas para nomear objetos.

A grande diferença entre a estrutura do XNA e as outras ferramentas de produção de jogos é a organização. Os métodos **Initialize, LoadContent, UnloadContent, Update** e **Draw** foram criados para ajudar você, programador, a estruturar melhor o seu jogo.

Para você incluir o código da classe Espaçonave, que foi visto na aula 4, dentro do seu projeto de jogo, basta colocá-lo logo abaixo da classe Game1, após a chave "}", que indica o fim da classe. Repare que será necessário adicionar também novos métodos dentro da classe Espaçonave para acessar e atualizar os valores dos atributos da classe, que são privados.

Para testar o código do seu projeto de jogo, selecione a opção **Build\Build Solution**, ou então aperte a tecla **F6**. Caso não obtenha sucesso, aparecerá uma listagem dos erros encontrados, bastando que você clique duas vezes sobre os erros listados para ir até o local onde está ocorrendo o problema. Os erros impedem o código de ser executado.

Fim do resumo

Informações sobre a próxima aula

Na próxima aula, veremos como exibir e movimentar a espaçonave do jogador com o Visual C#.