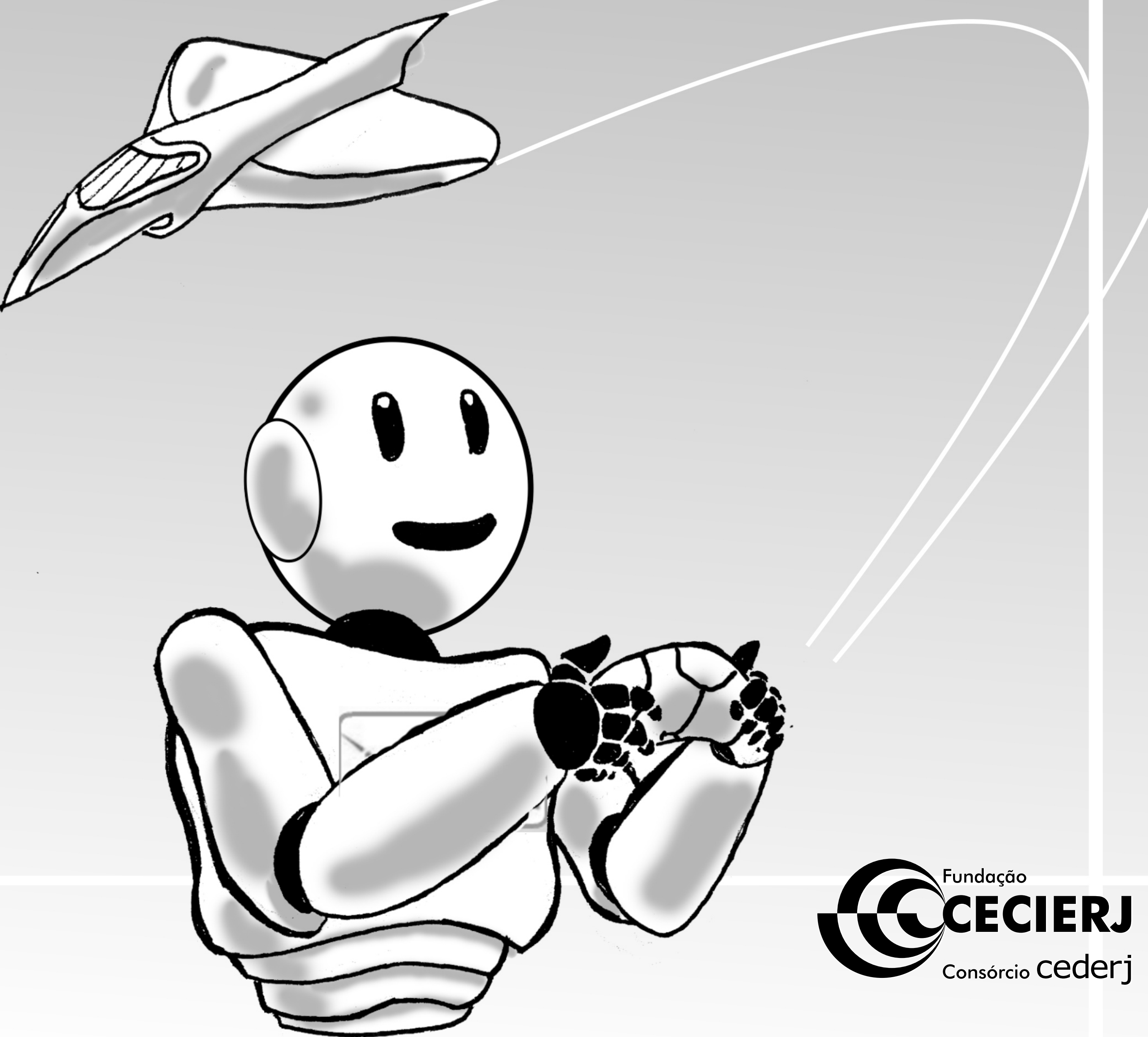
André Luiz Brazil
Lúcia Blondet Baruque

Volume único

Desenvolvendo Jogos 2D com C# e Microsoft XNA



Desenvolvendo Jogos 2D com C# e Microsoft XNA

Volume único

André Luiz Brazil
Lúcia Blondet Baruque

SECRETARIA DE
CIÊNCIA E TECNOLOGIA

Apoio:

Fundação Cecierj / Extensão

Rua Visconde de Niterói, 1364 – Mangueira – Rio de Janeiro, RJ – CEP 20943-001
Tel.: (21) 2334-1569 Fax: (21) 2568-0725

Presidente
Masako Oya Masuda

Vice-presidente
Mirian Crapez

Coordenadora da Área de Governança: Gestão, Auditoria e Tecnologia da Informação e Comunicação
Lúcia Blondet Baruque

Material Didático

ORGANIZAÇÃO
Lúcia Blondet Baruque

ELABORAÇÃO DE CONTEÚDO
André Luiz Brazil
Lúcia Blondet Baruque

REVISÃO LINGUÍSTICA
Alexandre Rodrigues Alves

Departamento de Produção

EDITORA
Tereza Queiroz

COPIDESQUE
Cristina Freixinho

REVISÃO TIPOGRÁFICA
Elaine Bayma
Daniela Souza

COORDENAÇÃO DE PRODUÇÃO
Katy Araújo

PROGRAMAÇÃO VISUAL
Ronaldo d´Aguiar Silva

ILUSTRAÇÃO
Jefferson Caçador

CAPA
Jefferson Caçador

PRODUÇÃO GRÁFICA
Patricia Seabra

Copyright © 2009, Fundação Cecierj / Consórcio Cederj

Nenhuma parte deste material poderá ser reproduzida, transmitida e gravada, por qualquer meio eletrônico, mecânico, por fotocópia e outros, sem a prévia autorização, por escrito, da Fundação.

B827d

Brazil, André Luiz.
Desenvolvendo Jogos 2D com C# e Microsoft XNA: volume único / André Luiz Brazil, Lúcia Blondet Baruque. – Rio de Janeiro: Fundação CECIERJ, 2010.
192p.; 19 x 26,5 cm.

ISBN: 978-85-7648-571-1

1. Jogos 2D. 2. Visual C#. 3. XNA Game Studio. I. Baruque, Lúcia Blondet. I. Título.

CDD: 006.6

2010/1

Referências Bibliográficas e catalogação na fonte, de acordo com as normas da ABNT e AACR2.

Governo do Estado do Rio de Janeiro

Governador
Sérgio Cabral Filho

Secretário de Estado de Ciência e Tecnologia
Alexandre Cardoso

Universidades Consorciadas

UENF - UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO
Reitor: Almy Junior Cordeiro de Carvalho

UERJ - UNIVERSIDADE DO ESTADO DO RIO DE JANEIRO
Reitor: Ricardo Vieiralves

UFF - UNIVERSIDADE FEDERAL FLUMINENSE
Reitor: Roberto de Souza Salles

UFRJ - UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Reitor: Aloísio Teixeira

UFRRJ - UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
Reitor: Ricardo Motta Miranda

UNIRIO - UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Reitora: Malvina Tania Tuttman

Desenvolvendo Jogos 2D com C# e Microsoft XNA

Volume único

SUMÁRIO

Prefácio ... 7

Introdução ... 11

Aula 1 – Conhecendo melhor os jogos ... 19

Aula 2 – Idealizando o seu jogo ... 37

Aula 3 – Instalando o Visual C# e o XNA Game Studio ... 51

Aula 4 – Criando a classe Espaçonave ... 61

Aula 5 – Iniciando o projeto de um jogo no Visual C# ... 75

Aula 6 – Exibindo e movimentando a sua espaçonave ... 95

Aula 7 – Acrescentando tiros e um cenário espacial ao seu jogo ... 115

Aula 8 – Acrescentando sons ao seu jogo ... 141

Aula 9 – Tratando colisões de objetos no jogo ... 159

Aula 10 – Produzindo efeitos duradouros (explosões) no jogo ... 173

Aula 11 – Tornando a espaçonave inimiga mais inteligente ... 181

prefácio

O tema jogos é muito importante hoje em dia na área de Informática. Não só pelo lado do entretenimento, como também porque propicia várias pesquisas nas áreas de interface de usuário e computação gráfica.

Quando eu ministrava aulas de interface de usuário, muitas vezes discutia com meus alunos as peculiaridades das interfaces para jogos, que deveriam ter um aspecto desafiador. Diferentemente das interfaces para outros aplicativos, que deveriam ter como características principais a facilidade de aprendizagem, de uso e de memorização, as interfaces para jogos geram um campo fértil para pesquisa nessa área, devido a seus aspectos específicos.

Na realidade, o impulso na área de jogos se verificou a partir do surgimento de novas plataformas para desenvolvimento de sistemas interativos. Essas plataformas ofereciam dispositivos de interação que iam muito além dos antigos dispositivos de ler e imprimir arquivos. Hoje, o usuário, com base em uma série de técnicas e estilos de interação, é imerso num mundo virtual, em várias áreas de aplicação, particularmente na de jogos, e interage de forma intuitiva, sem sofrer influência de tais dispositivos.

Por outro lado, o aspecto desafiador da interface de usuário de jogos leva à ideia de podermos aplicá-los em áreas empresariais – a partir da premissa de que as empresas podem jogar entre si, existindo inclusive os conceitos de jogos de empresas e jogos de guerra.

Por exemplo, antes de 1970 eu já era professor do ITA, e havia reuniões do Estado Maior da Aeronáutica em que se jogava com um programa chamado Jogo de Guerras. Recordo-me de que vinham vários oficiais para o ITA, que eram separados em grupo, e cada grupo representava um país. Os oficiais reuniam-se, faziam suas estratégias e entravam com parâmetros, perfurando cartões de leitura de forma a alimentar um programa de simulação para ver quem ganhava a guerra. Naquela época, usava-se como plataforma o computador IBM1130.

Nesse mesmo computador surgiu a possibilidade de conectar um terminal interativo, que tinha a capacidade de usar uma *light pen*. À época, desenvolvi vários programas de computação gráfica 2D para esse terminal, programando em Assembler e Fortran. Com essas subrotinas, fiz também um programa de animação na forma de jogos; era um programa que continha personagens ou caracteres e tinha um *clock* que permitia alterar a posição do personagem na tela. Criei também um jogo em que dois aviões travavam uma batalha em cima de um aeroporto. A montanha e o aeroporto eram caracteres fixos. O avião se movimentava e tinha um piloto. Um avião podia se atirar contra o outro e existia ainda a possibilidade de o piloto pular de paraquedas. O programa servia de brincadeira para os alunos do curso do ITA, enquanto o jogo sério era jogado pelos coronéis. Isso era algo que requeria alguma técnica para movimentar o avião e o personagem.

A partir da década de 80, surgiram programas no APPLE II, cujo código era feito em Basic, tipo Packman e Ping-Pong. O Apple já permitia uma tela interativa e o uso do *mouse*.

Pensando em jogos de guerra, em um nível mais alto, faz sentido relacionar a área de jogos à de governança. As técnicas de jogos podem ser aplicadas na área de gestão. Hoje em dia fala-se muito em inteligência competitiva nessa área. A inteligência competitiva visa a gerenciar o conhecimento externo à empresa para melhorar o processo decisório interno. Isso pode ser visto como um jogo. Então, a ideia de jogos, que em princípio está atrelada a entretenimento, torna-se uma ideia de jogo real.

Este livro de jogos vem no momento certo, trazendo técnicas e conceitos que são ensinados de forma lúdica, e que poderão mais tarde ser utilizados em diversos níveis estratégicos da empresa.

Didaticamente falando, o livro possui uma linguagem bastante intuitiva e de fácil aprendizado. Destina-se principalmente a iniciantes ou curiosos que estejam interessados em saber mais sobre como se produz um jogo 2D. O livro assume que o leitor já possui algum conhecimento de lógica de programação e o ensina a programar jogos na linguagem C#.

O livro exemplifica a construção de um jogo espacial. É completo e aborda desde a origem e história da evolução dos jogos e o panorama atual dos jogos na educação até conceitos mais rebuscados, como a preparação de um plano para a construção do jogo, orientação a objetos, exibição de texturas, movimentação de personagens, incorporação de sons ao jogo, colisões entre objetos do jogo e inteligência artificial.

Ao ler o livro, o leitor também aprenderá a instalar e configurar o Visual Studio – o ambiente que permite a programação em C#. Ao longo de cada aula, são encontrados diversos exemplos e atividades práticas para exercitar e consolidar o aprendizado.

Rubens Nascimento Melo

Rubens Nascimento Melo possui graduação em Engenharia Eletrônica pelo Instituto Tecnológico de Aeronáutica, mestrado e doutorado em Ciência da Computação também pelo Instituto Tecnológico de Aeronáutica. Atualmente, é professor associado da Pontifícia Universidade Católica do Rio de Janeiro, onde leciona cursos de graduação e pós-graduação, orienta alunos de iniciação científica, mestrado e doutorado. Tem mais de 40 anos de experiência na área de Ciência da Computação em geral e é um dos pioneiros nas áreas de Banco de Dados, Computação Gráfica e Interface de Usuário no Brasil. Seus temas de pesquisa principais são: integração semântica de informação, *data warehousing*, *Business Intelligence*, *e-learning* e sistemas de informação.

introdução

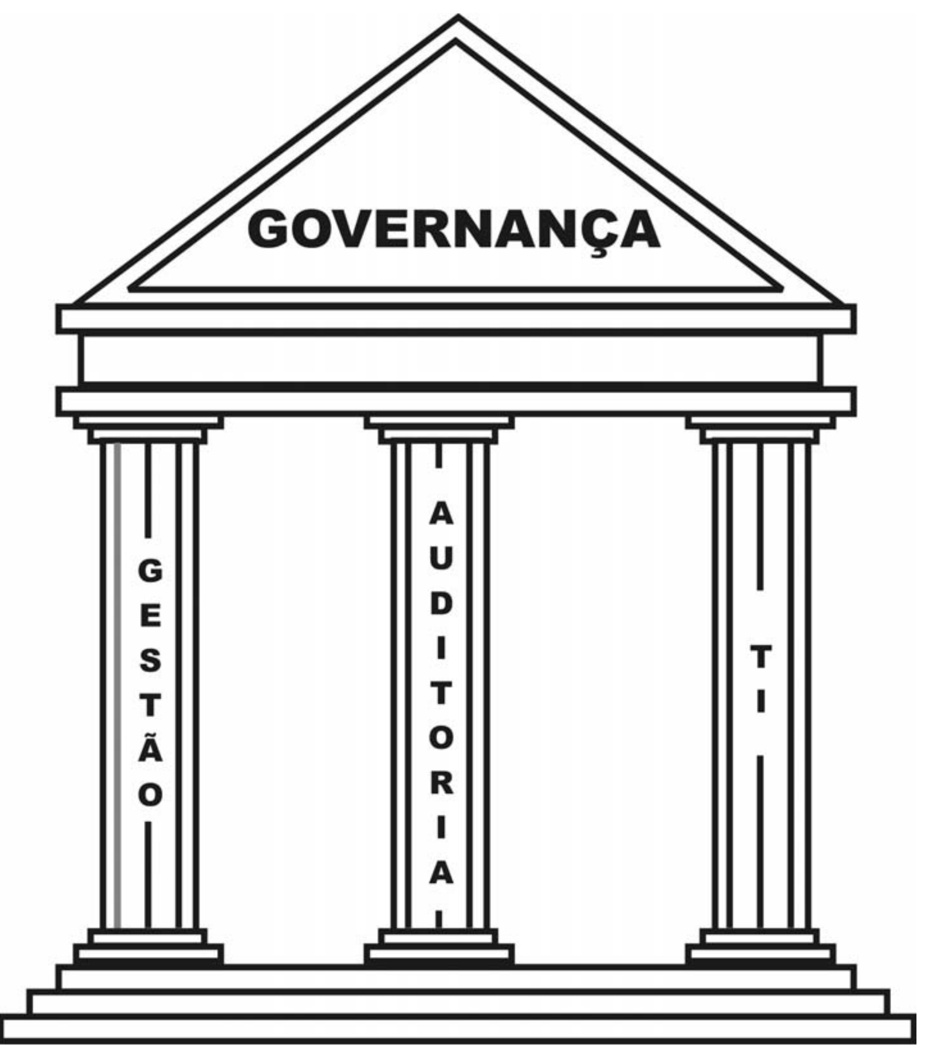
Você sabe o que é governança?

Governança é o conjunto de responsabilidades e práticas exercidas pela diretoria e pela gerência executiva com o objetivo de prover uma direção estratégica à empresa, assegurando que seus objetivos sejam alcançados e seus riscos gerenciados apropriadamente, verificando que seus recursos sejam usados de forma responsável, com ética e transparência.

Repare que, em linhas gerais, o processo de governança nas empresas visa a responder a quatro questões básicas:

1. Se a empresa está fazendo as coisas certas;
2. Se a empresa está atuando de forma correta;
3. Se o uso de recursos é eficaz e eficiente;
4. Se os objetivos estabelecidos são alcançados.

Observe que o conceito de governança é relativamente novo, mas já se reconhece que boas práticas de governança aplicam-se a qualquer tipo de empreendimento. Pense e responda: quais são as três principais áreas do conhecimento que podem contribuir diretamente para uma boa governança?



Cada uma dessas áreas tem um objetivo definido dentro da governança:

- Gestão – estabelece um sistema de **controle** gerencial, bem como um ambiente que promova o alcance dos objetivos do negócio;
- Auditoria – avalia de forma independente a adequação e a eficácia dos **controles** estabelecidos pela gerência/diretoria;
- Tecnologia da Informação e Comunicação – apoia e capacita a execução dos **controles** do nível estratégico ao operacional.

STAKEHOLDERS

São aquelas pessoas ou instituições que possuem algum tipo de envolvimento profissional ou pessoal com uma empresa: investidores, clientes, funcionários, fornecedores, credores, acionistas, usuários, parceiros etc.

A **governança corporativa** tornou-se um tema dominante nos negócios por ocasião dos vários escândalos financeiros ocorridos nos EUA em meados de 2002 – Enron, Worldcom e Tyco, para citar apenas alguns. A gravidade de tais escândalos abalou a confiança de investidores, realçando a necessidade das empresas de proteger o interesse de seus STAKEHOLDERS. A governança corporativa tem a gerência de risco como um de seus principais componentes, que são: planejamento estratégico, liderança, definição de processos, acompanhamento e gerência de riscos.

O papel dos jogos na governança

Sociedade da Informação é aquela cujo modelo de desenvolvimento social e econômico baseia-se na informação como meio de criação do conhecimento. Ela desempenha papel fundamental na produção de riqueza e na contribuição para o bem-estar e a qualidade de vida dos cidadãos. Tal sociedade encontra-se em processo de formação e expansão.

Na sociedade da informação ou do conhecimento, indivíduos como você precisam estar aptos a lidar com seu novo papel com agilidade e maior autonomia frente às diversas situações, o que gera a necessidade de prover tais indivíduos com o máximo de conhecimento. A sociedade atual requer que pessoas como você desenvolvam as competências críticas dessa nova era.

Os jogos podem ser uma excelente ferramenta na capacitação dos **profissionais do conhecimento**, incluindo até os executivos de alto nível, que são os maiores responsáveis pela governança. Trata-se de uma abordagem altamente motivadora, na qual os aprendizes seguem regras para atingir determinado desafio.

Profissionais do conhecimento é um termo adaptado de *knowledge* worker, referindo-se a profissionais que trabalham diretamente com a informação, produzindo ou fazendo uso do conhecimento, em oposição àqueles envolvidos na produção de bens ou serviços. São analistas de sistemas, programadores, desenvolvedores de produtos ou pessoal ligado à tarefa de planejamento e análise, bem como pesquisadores envolvidos principalmente com a aquisição, análise e manipulação da informação.
Esse termo foi popularizado pelo guru Peter Drucker.

Os jogos podem ser usados em várias situações, com o objetivo de aumentar a motivação e o nível de esforço, promovendo:

- Interação e comunicação entre os pares e o exercício da competitividade.
- Oportunidade para a prática de habilidades com *feedback* imediato.
- Ajuda para lidar com situações imprevisíveis.
- Melhoria nas habilidades de tomada de decisão.
- Auxílio na aprendizagem de conceitos básicos.
- Desenvolvimento de habilidades de liderança, cooperação e trabalho em equipe.

Perceba que muitas das habilidades críticas exigidas na Era do Conhecimento, a serem desenvolvidas por você, profissional do conhecimento, podem ser obtidas por intermédio dos jogos, que possuem ainda a vantagem de promover um ambiente lúdico e prazeroso para o aprendizado.

A Fundação Cecierj trabalha para apoiar você nessa meta. Representada pela professora Masako Masuda e pela professora Mirian Crapez, a Fundação decidiu ampliar a sua missão e incluir no escopo do seu público-alvo os profissionais de mercado. No início de 2008, foi estabelecida uma nova área do conhecimento na Diretoria de Extensão, voltada para a oferta de cursos não só aos professores da Educação Básica, mas também aos profissionais do mercado, intitulada Governança: Gestão, Auditoria e Tecnologia da Informação e Comunicação (TIC).

Um dos objetivos dessa área, coordenada pela professora Lúcia Baruque, é de que os profissionais do conhecimento, que são parte essencial da sociedade da informação, possam se capacitar e se atualizar em temas de ponta, incluindo aqueles que moram no interior do estado.

Masako Oya Masuda

Presidente da Fundação Centro de Ciências e Educação Superior a Distância do Estado do Rio de Janeiro (Cecierj), vinculada à Secretaria de Estado de Ciência e Tecnologia. Possui graduação em Ciências Biológicas pela Universidade de São Paulo, mestrado em Ciências Biológicas (Biofísica) pela Universidade Federal do Rio de Janeiro, doutorado em Ciências Biológicas (Biofísica) pela Universidade Federal do Rio de Janeiro e pós-doutorado na University of California. Tem experiência na área de Fisiologia, com ênfase em Fisiologia Geral.

Mirian Araujo Carlos Crapez

Vice-presidente de Educação Superior a Distância da Fundação Cecierj. Graduada em Ciências Biológicas pela Universidade Federal de Minas Gerais, com doutorado em Metabolismo de Aromáticos e Poliaromáticos realizado na Université D'Aix-Marseille II e pós-doutorado na Université Paris VI (Pierre et Marie Curie). Atualmente, é professora associada da Universidade Federal Fluminense.

Lúcia Blondet Baruque

Mestre e doutora em Informática pela PUC-Rio; bacharel em Ciências Econômicas pela UFRJ; possui *Certificate in Management* pela John Cabot University (Roma). É professora associada da Fundação Cecierj e pesquisadora associada do Laboratório de Tecnologia em Banco de Dados da PUC-Rio, tendo trabalhado na CEAD desta instituição. Atuou na auditoria da Exxon Company International e na ONU, em Roma, bem como na diretoria do Instituto dos Auditores Internos do Brasil. Membro do Institute of Internal Auditors. CIA, CISA *Exam*.

André Luiz Brazil

Mestre em Computação Visual e Interfaces pela Universidade Federal Fluminense (UFF) e pós-graduado em Análise, Projeto e Gerência de Sistemas na PUC-Rio. Analista de sistemas de informação da Fundação Getulio Vargas com experiência em programação nas linguagens XNA, C#, Asp.Net, Sql, C++, OpenGL, SDL, Cold Fusion, Visual Basic e Delphi. Apaixonado por *games* e pelo ensino, tem como ideal o desenvolvimento de jogos voltados para a educação.

O curso Desenvolvendo Jogos 2D com C# e XNA, desenvolvido pelo professor André Luiz Brazil, possui os seguintes objetivos: capacitar o participante no processo de criação, estruturação e funcionamento de um jogo 2D; apresentar fundamentos básicos sobre programação na linguagem C#; introduzir os conceitos de orientação a objeto e inteligência artificial.

Ementa

• Introdução ao desenvolvimento de jogos
• Instalação do XNA Game Studio
• Criação de projetos no Visual Studio
• Noções de orientação a objeto
• Criação de jogo espacial em 2D
• Tratamento de colisões entre objetos
• Noções de inteligência artificial

Carga horária: 45 horas/aula

Agora que você está familiarizado com o conceito de governança e com os requisitos de aprendizagem desta era em que vivemos, mãos à obra. Vamos aprender muito sobre jogos a seguir.

Bons estudos e conte conosco!

Referências Bibliográficas

BARUQUE, L. *ELGORM*: um modelo de referência para governança de *e-learning*. 2004. Tese (Doutorado)–Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 2004.

HAMAKER S.; HUTTON A. Principles of Governance. *Information Systems Audit and Control Journal*. v. 3, 2003.

TERRA J. C. C. *Gestão do conhecimento*: o grande desafio empresarial. 2. ed. São Paulo: Negócio, 2001.

AULA 1

Conhecendo melhor os jogos

Meta da aula

Apresentar as características dos jogos e suas categorias de aprendizado.

objetivos

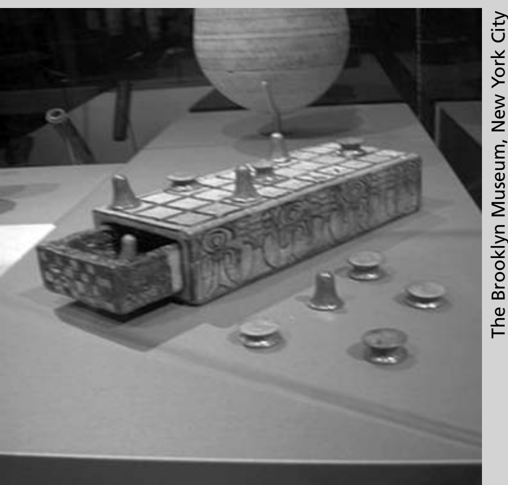
Ao final desta aula, você deverá ser capaz de:

1. identificar as características necessárias em um jogo;
2. conceituar categoria de aprendizado;
3. classificar um jogo em uma ou mais categorias de aprendizado;
4. identificar as possíveis vantagens do uso de jogos na educação.

INTRODUÇÃO

Você gosta de jogos? Que bom! Eles existem por toda parte. Os mais comuns hoje em dia são os jogos de tabuleiro e os de videogame, mas temos também diversos outros tipos, como os jogos olímpicos, por exemplo.

Você sabia que o jogo de tabuleiro mais antigo surgiu em 3500 a.C.?
O nome do jogo era Senet. Dizem que era um jogo de corrida para dois jogadores com um tabuleiro de trinta casas.
O jogo foi encontrado nas tumbas do antigo Egito e simbolizava uma espécie de talismã para ajudar a enfrentar a jornada pós-vida, devido ao elemento sorte e à crença egípcia no determinismo.



The Brooklyn Museum, New York City

Figura 1.1: Senet, o jogo de tabuleiro mais antigo, de 3500 a. C.
Fonte: http://upload.wikimedia.org/wikipedia/commons/thumb/f/fb/P9210016.JPG/300px-P9210016.JPG

Em 2006 as empresas produtoras de jogos eletrônicos faturaram cerca de 9,5 bilhões de dólares, quase atingindo o faturamento anual da indústria do cinema. Veja a seguir dois exemplos de consoles de videogame: o Atari, lançado em 1977, e o XBOX 360, lançado em 2006.

Atari Inc.

Figura 1.2: Atari 2600, lançado em 1977.
Fonte: http://upload.wikimedia.org/wikipedia/commons/thumb/d/dc/Atari2600a.JPG/300px-Atari2600a.JPG

Microsoft

Figura 1.3: XBOX 360, lançado em 2006.
Fonte: http://upload.wikimedia.org/wikipedia/commons/thumb/6/69/Xbox360.png/250px-Xbox360.png

CARACTERÍSTICAS DOS JOGOS

Pense em algum jogo que você já jogou. Tente agora definir o que você acha que caracteriza um jogo. Chris Crawford, grande produtor de jogos e escritor, relacionou quatro características necessárias e que precisam estar presentes em um jogo:

- Representação: o jogo apresenta de forma subjetiva um subconjunto da realidade.
- Interação: o jogador faz as coisas acontecerem, modifica o ambiente e os resultados.
- Conflito: a presença constante de desafios e objetivos dentro do jogo testa a habilidade do jogador.
- Segurança: os resultados de um jogo são sempre menos dramáticos que as situações reais.

Jogar geralmente é divertido ou, pelo menos, deveria ser. De que tipo de jogo você gosta mais: Palavras cruzadas, xadrez, gamão ou jogos eletrônicos?

Atividade de reflexão

Pense agora num jogo de que você gosta e indique quais características, dentre as citadas, você acha que se encontram presentes nesse jogo. Explique também por que essas características estão presentes no jogo.

____
____
____
____
____
____

CATEGORIAS DE APRENDIZADO DOS JOGOS

Provavelmente você já deve ter aprendido algo interessante ao jogar. Pense em quantas coisas bacanas você já aprendeu simplesmente por estar jogando o jogo da memória ou o de palavras cruzadas. Ao pensar no que você aprendeu ao jogar um jogo, você define um conceito importante, que é a **CATEGORIA DE APRENDIZADO** desse jogo.

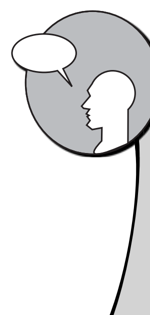
> **CATEGORIA DE APRENDIZADO**
> É classificação de um jogo de acordo com o tipo de conhecimento que ele tem a oferecer para o jogador.

Os jogos podem ensinar ou exercitar diferentes habilidades do jogador. Eles costumam pertencer a uma ou mais categorias de aprendizado. Pense num jogo de xadrez, por exemplo. Que tipo de conhecimento ou habilidade ele tem a oferecer para você, como jogador?

Podemos dizer que o xadrez estimula o raciocínio e a memória. Verifique em qual das três categorias de aprendizado a seguir você acha que um jogo de xadrez feito para o computador se enquadraria:

- *adventures/quest games*;
- jogos de labirinto;
- jogos tradicionais.

Sim, o xadrez feito para computador se enquadra na categoria de aprendizado “jogos tradicionais”, ou seja, jogos que foram produzidos inicialmente em tabuleiro e que depois foram desenvolvidos para o computador. Os jogos pertencentes a essa categoria de aprendizado estimulam o raciocínio e a memória do jogador.



Atividade comentada 1

Agora tente selecionar outros jogos que fazem parte da categoria de aprendizado "jogos tradicionais":

( ) Jogo da memória
( ) Jogo de gamão
( ) Counter-strike (CS)
( ) Paciência

Vamos ver agora uma lista mais completa de categorias de aprendizado para os jogos eletrônicos e definir com mais detalhes as características de cada uma delas:

- *adventures/quest games*;
- simulações;
- jogos de corrida;
- jogos de labirinto;
- atividades de "*edutainment*", ou entretenimento educativo;
- criativos/construção de modelos;
- jogos de tiro/arcade;
- jogos tradicionais.

Adventures/quest games

Pense numa aventura na qual você precisa explorar uma caverna escondida no subterrâneo de uma grande ilha, com vários enigmas a serem desvendados e armadilhas a serem ultrapassadas, uma a uma, até encontrar um grande tesouro que foi escondido por piratas muito tempo atrás. Os jogos dessa categoria de aprendizado são assim:

- Apresentam uma sequência de passos ou desafios a serem atingidos pelo jogador, dentro do jogo.
- Normalmente possuem um cenário bem específico. Pode ser uma cidade ou local histórico ou um ambiente fictício, por exemplo.
- As tarefas nesse tipo de jogo podem ser relevantes para o currículo escolar.

- Conhecimentos sobre a história ou geografia de um local, por exemplo, podem ser tipicamente encaixados ou obtidos nesse tipo de jogo.

Nemesis informática

Figura 1.4: Gruta de Maquiné (1989), jogo de *adventure* brasileiro.
Fonte: http://bertelli.name/stuffs/misc/2007/11/grutademaquine.png

Lucasfilm Games

Figura 1.5: The Secret of Monkey Island (1990).
Fonte: http://www.worldofmi.com/images/categories/2/screen03.gif

Ubisoft Montreal

Figura 1.6: Prince of Persia: The sands of time (2003).
Fonte: http://image.com.com/gamespot/images/2003/news/05/09/prince/prince_screen007.jpg

Simulações

Imagine agora que você é o dono de uma imobiliária e que precisa administrar a compra e venda de imóveis, negociando preços, terrenos e tudo mais. E ainda por cima, você tem de lidar com as imobiliárias concorrentes e tentar fechar o mês no azul. Os jogos dessa categoria de aprendizado são mais ou menos assim:

- O jogador interage com um ambiente que é uma simulação de uma estrutura ou modelo existente na vida real. Também pode simular outros modelos, sendo eles futuristas ou, ainda, imaginários.

- Conhecimentos sobre administração, estratégia ou habilidades específicas podem ser tipicamente encaixados ou obtidos com esse tipo de jogo.

Ensemble Studios

Figura 1.7: Age of Empires (1997).
Fonte: http://image.com.com/gamespot/images/screenshots/0/90380/ageof_screen002.jpg

Mumbo Jumbo

Figura 1.8: Lemonade Tycoon II (2004).
Fonte: http://image.com.com/gamespot/images/2004/screen0/922248_20040810_screen002.jpg

Microsoft Game Studios

Microsoft Game Studios

Figuras 1.9 e 1.10: Microsoft Flight Simulator X (2006). Simulador de vôo para treinar pilotos.
Fonte: http://image.com.com/gamespot/images/2006/284/reviews/931252_20061012_screen001.jpg
Fonte: http://image.com.com/gamespot/images/2006/288/reviews/931252_20061016_screen010.jpg

Jogos de corrida

Você é um ás do volante e compete com diversos outros oponentes num grande rali, podendo consertar e fazer ajustes no seu carro a cada etapa do percurso, de acordo com o terreno a ser percorrido. Os jogos dessa categoria de aprendizado possuem as seguintes características:

- O jogador dirige um veículo percorrendo um trajeto específico, geralmente com obstáculos e outros oponentes.
- Em alguns jogos desse tipo, é possível criar novos trajetos e/ou configurar os veículos.

- Conhecimentos criativos ou sobre habilidades específicas podem ser tipicamente encaixados ou obtidos com esse tipo de jogo.

Sphere Inc.

Figura 1.11: Stunt Driver (1990).
Fonte: http://upload.wikimedia.org/wikipedia/en/thumb/0/01/Stunt_Driver_splash_screen.jpg/250px-Stunt_Driver_splash_screen.jpg

EA Black Box, EA UK

Figura 1.12: Need for Speed Carbon (2006).
Fonte: http://image.com.com/gamespot/images/2006/325/934371_20061122_screen001.jpg

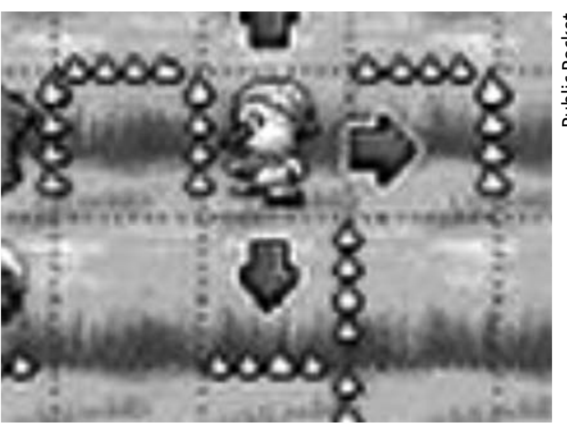
Jogos de labirinto

Quem nunca jogou o jogo do PAC-man, controlando aquele bichinho amarelo que percorria o labirinto e que, para passar de fase no jogo, tinha de comer todas as pastilhas brancas com a boca e ao mesmo tempo fugir dos fantasminhas coloridos, que iam ficando cada vez mais rápidos? Assim como no PAC-man, os jogos dessa categoria de aprendizado possuem:

- movimentação em espaço bidimensional ou tridimensional, com obstáculos, objetivos ou fases a atingir;
- possíveis desafios relativos a tempo, habilidades de memória e/ou motores e planejamento;
- conhecimentos de estratégia, organização, raciocínio rápido e tomada de decisões sob pressão podem ser tipicamente encaixados nesse tipo de jogo.

Hiroyuki Imabayashi

Figura 1.13: Sokoban (1980).
Fonte: http://upload.wikimedia.org/wikipedia/en/thumb/9/92/Sokoban_PC_game_01.png/180px-Sokoban_PC_game_01.png



Public Pocket

Figura 1.14: Ice Maze (2004).
Fonte: http://image.com.com/gamespot/images/2004/reviews/922041_20040730_screen004.jpg

Atividades de "*edutainment*", ou entretenimento educativo

Imagine um jogo parecido com uma revista, daquelas que você encontra à venda nas bancas de jornal, em que você tem escritas algumas palavras cruzadas entremeadas e vários quadradinhos em branco para completar as outras palavras. Algumas dicas ou pistas lhe ajudam a lembrar e completar as outras palavras, estimulando a sua memória. Jogos dessa categoria são:

- estruturados com foco em suporte educacional;
- aplicáveis em desenvolvimento de habilidades como coordenação motora, concentração, memória, resolução de problemas ou criação de ambiente facilitador de aprendizado baseado em conteúdo como vídeos, clipes, figuras ou qualquer material relacionado ao foco do aprendizado.

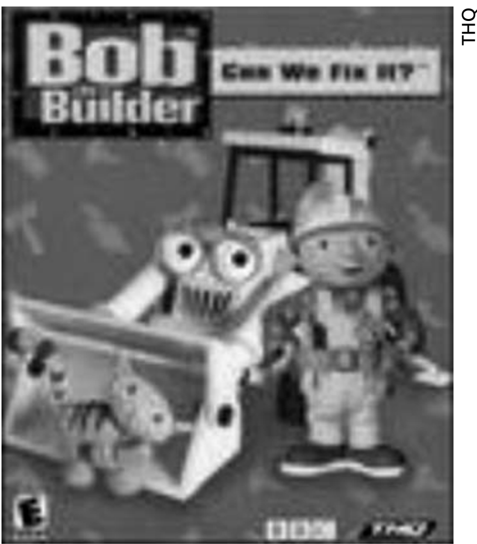
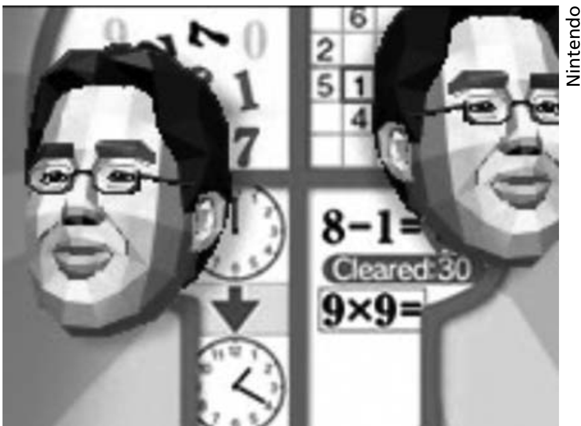


Figura 1.15: Bob The Builder (2001).
Fonte: http://image.com.com/gamespot/images/2003/all/boxshots2/929902_70056.jpg



Nintendo

Figura 1.16: Brain Age (2006).
Fonte: http://image.com.com/gamespot/images/titles/product/7/931667.jpg

Criativos/construção de modelos

Você já ouviu falar do brinquedo lego? Visualize um jogo de corrida no qual você pode construir com as peças o seu próprio motorista, montar o carro dele encaixando rodas, cano de descarga, vidros e tudo mais. Os jogos que pertencem a essa categoria apresentam:

- possibilidade de construir, alterar, modificar ou configurar o ambiente e/ou os personagens do jogo;
- possibilidade de estimular a criatividade, e conhecimentos de planejamento e organização podem ser tipicamente encaixados nesse tipo de jogo.

High Voltage Software

Figura 1.17: Lego Racers (1999).
Fonte: http://image.com.com/gamespot/images/screenshots/4/197774/legorace_screen003.jpg

Chris Sawyer

Figura 1.18: Roller Coaster Tycoon (1999).
Fonte: http://image.com.com/gamespot/images/titles/product/6/132866.jpg

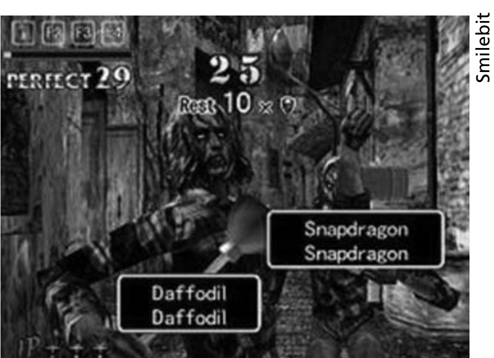
Jogos de tiro/arcade

Já pensou num jogo em que você é um agente policial e sua missão é perseguir grandes bandidos da Máfia, entrar sorrateiramente em seus esconderijos e com a precisão de um atirador de elite dar cabo de todos eles? Os jogos dessa categoria:

- normalmente envolvem mirar e atirar em objetos e criaturas em movimento, ferindo ou destruindo-as. Podem conter objetivos ou etapas específicas ou opcionais dentro do cenário de jogo;
- estimulam o desenvolvimento da coordenação motora e o raciocínio rápido. Conhecimentos de táticas de guerra e trabalho em equipe também podem ser tipicamente encaixados nesse tipo de jogo.

Raven Software

Figura 1.19: Soldier of Fortune (1999).
Fonte: http://image.com.com/gamespot/images/screenshots/9/198689/sof_screen001.jpg



Smilebit

Figura 1.20: Typing of the Dead (1999), para matar os zumbis o jogador precisa digitar – rapidamente!
Fonte: http://upload.wikimedia.org/wikipedia/en/thumb/9/9c/Typingdeadgameplay.jpg/250px-Typingdeadgameplay.jpg

EALA

Figura 1.21: Medal of Honor: Airborne (2007).
Fonte: http://image.com.com/gamespot/images/2007/247/931484_20070905_screen004.jpg

Jogos tradicionais

- Normalmente, jogos de tabuleiro ou de cartas com sua respectiva versão eletrônica. Costumam fazer bom uso de inteligência artificial para simular oponentes competitivos.
- Estimulam o desenvolvimento de habilidades ou conhecimentos variados. Estímulos à memória e ao raciocínio podem ser tipicamente encaixados nesse tipo de jogo.

Interplay

Figura 1.22: Battle Chess (1988): o tabuleiro de xadrez vira uma batalha medieval.
Fonte: http://www.lauppert.ws/screen1/bchess.png

Interplay

Figura 1.23: ChessMaster 5000 (1996).
Fonte: http://image.com.com/gamespot/images/screenshots/6/196906/ches5000_screen002.jpg

Leaping Lizard Software

Figura 1.24: Magic: The Gathering Online (2002).
Fonte: http://image.com.com/gamespot/images/2002/pc/str/magic/magic_screen001.jpg

Atividade 1

Agora que você já aprendeu um pouco sobre as categorias de aprendizado dos jogos, abra o seu navegador e procure na internet um exemplo de jogo para cada uma das categorias de aprendizado exibidas.
Dica: experimente utilizar *sites* de informações sobre jogos, tais como www.gamespot.com.

Registre aqui as suas descobertas:
____
____
____
____
____
____
____
____
____
____

O uso dos jogos na educação

Você acha que a educação supostamente também vem evoluindo com o uso da tecnologia?

Com a evolução da tecnologia, temos o aumento do acesso à informação, principalmente com o aparecimento da televisão e da internet. Contudo, o uso da tecnologia na educação permanece questionável.

Entramos na era do copiar e colar, em que os trabalhos escolares são muitas vezes feitos copiando-se textos prontos, sem entendê-los. Atualmente, o ambiente escolar já não é considerado um dos mais prazerosos para a grande maioria dos alunos.

Como você acha que os jogos podem apoiar as atividades educativas?

- Por serem considerados uma atividade lúdica e divertida, configuram-se como uma boa opção para tornar o ambiente escolar mais agradável e atrativo.

- Podem também reduzir ou eliminar os bloqueios presentes no aprendizado tradicional, se conduzidos de forma correta pelo professor.
- São interativos e estimulam a atenção e a concentração dos alunos, exigindo que estes executem ações e tomem decisões para atingir seus objetivos.

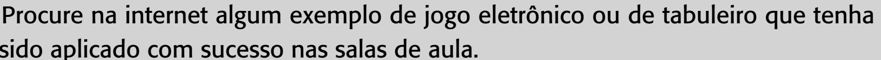
O cenário atual dos jogos na educação

A maior parte das empresas desenvolvedoras de jogos não se preocupa necessariamente em criar jogos voltados para a educação. As grandes empresas, em geral, têm medo de "arriscar" o seu nome em projetos de jogos educativos. Boa parte dos jogos criados com o rótulo "educativo" não é projetada adequadamente e acaba por deixar de ser interessante ou cativante.

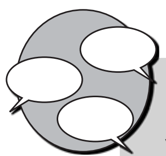
Quais soluções você acha que seriam boas para tentar contornar esse problema?

- Subdividir os jogos existentes em categorias de aprendizado conforme visto antes agrupando-os e classificando-os conforme o tipo de conhecimento que têm a oferecer.
- Desenvolver jogos com foco em um objetivo bastante específico, como treinar determinada habilidade ou prover conhecimento sobre uma situação, época ou tópico.

Atividade 2

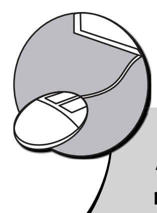


Procure na internet algum exemplo de jogo eletrônico ou de tabuleiro que tenha sido aplicado com sucesso nas salas de aula.



INFORMAÇÕES SOBRE FÓRUM

Você também é apaixonado por jogos? Então, vamos nos conhecer e trocar informações no fórum da semana.



Atividade *online*

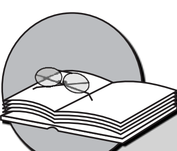
Agora que você já conhece um pouco melhor os jogos, então vá à sala de aula virtual e resolva a atividade proposta pelo tutor.

RESUMO

Os bons jogos apresentam quatro características necessárias e bem definidas: representação, interação, conflito e segurança. Podem ser subdivididos em categorias de aprendizado, ou seja, de acordo com o que têm a oferecer ao jogador, em termos de conhecimento.

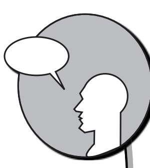
As categorias de conhecimento mais presentes nos jogos de hoje são: *adventures* / *quest games*, simulações, jogos de corrida, jogos de labirinto, atividades de "*edutainment*" ou entretenimento educativo, criativos / construção de modelos e jogos de tiro/arcade.

Atualmente, a maior parte das empresas desenvolvedoras de jogos não se preocupa necessariamente em criar jogos voltados para a educação. Para solucionar esse problema, podemos subdividir os jogos já existentes em categorias de aprendizado ou desenvolver jogos com foco em um objetivo educacional bastante específico.



Informação sobre a próxima aula

Na próxima aula, veremos quais ferramentas utilizaremos para desenvolver os jogos e quais são os profissionais envolvidos na produção de um jogo.



Resposta da atividade comentada

1

(x) Jogo da memória

(x) Jogo de gamão

(   ) Counter-strike (CS)

(x) Paciência

Dos quatro jogos já apresentados, o Counter-strike (CS) é o único que não faz parte da categoria de aprendizado "jogos tradicionais". Como ele é um jogo de tiro, pertence à categoria de aprendizado "jogos de tiro/arcade".

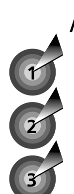
Idealizando o seu jogo

Meta da aula

Apresentar idéias e ferramentas para a criação dos jogos.

objetivos

Ao final desta aula, você deverá ser capaz de:



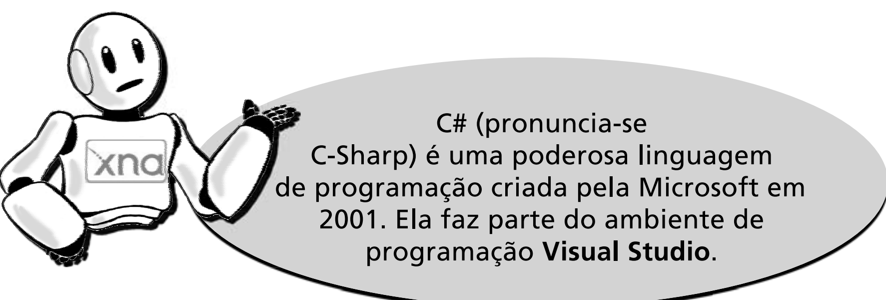
1. conceituar C#, Visual Studio e XNA;
2. idealizar um jogo;
3. descrever o papel dos profissionais envolvidos na criação de um jogo.

Pré-requisitos

Conhecer e identificar as categorias de aprendizado, conceito abordado na Aula 1.

INTRODUÇÃO

Pois bem, agora que você já conhece um pouco mais sobre jogos, vai aprender um pouco também sobre as ferramentas que irá utilizar para desenvolver seu próprio jogo, além dos diversos profissionais envolvidos na criação desses jogos.
A esta altura, você deve estar se perguntando o que significam essas duas siglas esquisitas, **C#** e **XNA**. Pois bem, veja a explicação:



Você sabia que a linguagem **C#** tem suas origens baseadas na linguagem C, que foi criada em 1972 por Dennis Ritchie, um estudante de Física e Matemática da Universidade de Harvard?
Os sistemas operacionais atuais Windows e Linux (isso mesmo, os que você usa hoje!) foram produzidos com o C++, que é uma versão evoluída da linguagem C.

Como fazer então para criar um jogo utilizando a linguagem **C#**?

Existe um "ajudante", muito competente por sinal, que vai indicando tudo o que você escreveu errado no seu programa e dando dicas para você fazer as correções. Esse "ajudante" é o **Visual Studio**.

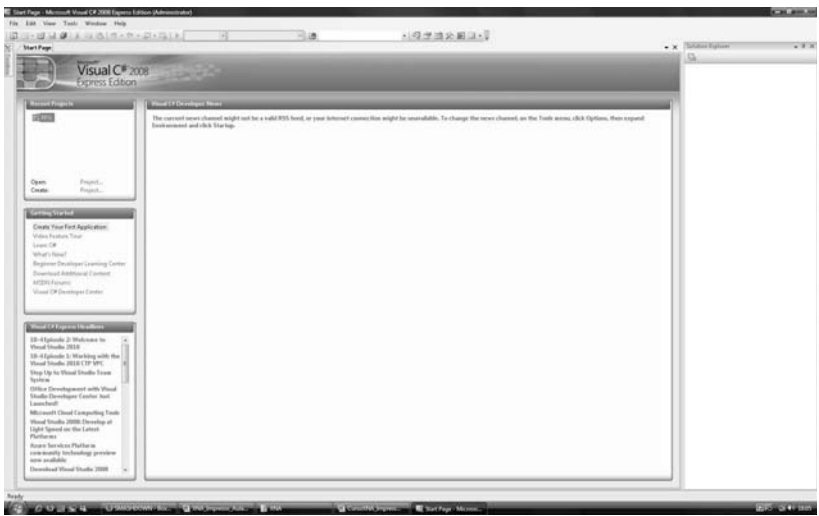
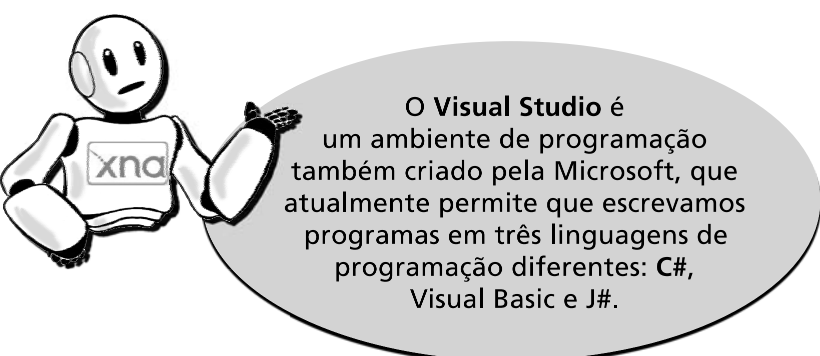


Figura 2.1: Tela inicial do Visual Studio.
Fonte: Aplicativo Visual Studio



O **Visual Studio** costuma chamar um grande "amigo" quando vai produzir algo mais complicado, como um jogo. Esse "amigo" é muito esperto e organizado, e consegue se comunicar mais facilmente com outras partes do computador, como vídeo e som, por exemplo. O nome dele é **XNA**.

O XNA, que em inglês significa **N**e**x**t Ge**n**eration **A**rchitecture, ou a arquitetura da próxima geração, é uma estrutura de desenvolvimento (FRAMEWORK) que possui um conjunto de funções específicas para a produção de jogos. Sua primeira versão foi lançada pela Microsoft no final de 2005.

FRAMEWORK

É uma estrutura de organização criada para padronizar e facilitar o desenvolvimento de um programa.

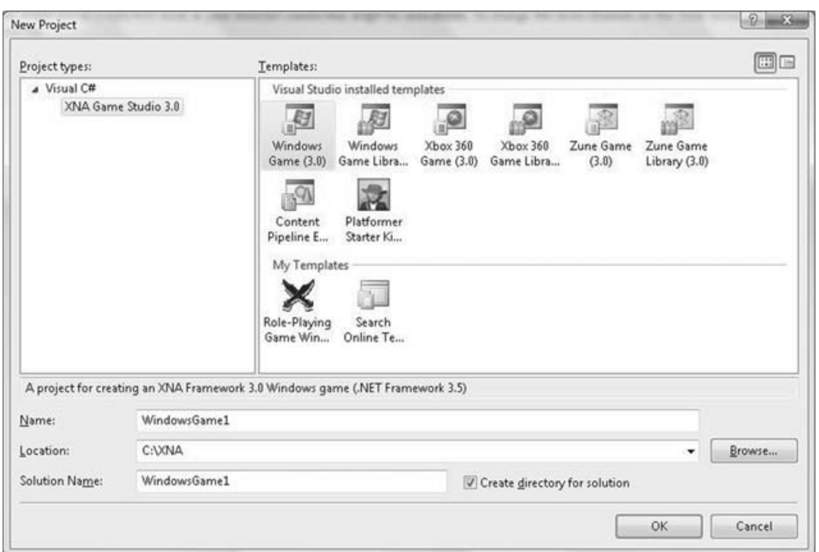


Figura 2.2: Iniciando um projeto de jogo em XNA.
Fonte: Aplicativo Visual Studio

Repare que a preocupação principal do **XNA** é encapsular (guardar para si) as funções e os detalhes técnicos de cada tipo de computador.

Dessa forma, você, que irá desenvolver o jogo, não precisará se preocupar muito com isso e poderá se concentrar mais na produção do conteúdo e da mecânica do jogo em si. Essa é a grande vantagem do **XNA** em relação a outras ferramentas de desenvolvimento de jogos, que requerem um nível de conhecimento muito maior para permitir a produção de um jogo completo.

Quer um exemplo? Imagine que você deseja desenhar um boneco na tela do

seu jogo. Basta dizer para o **XNA** em qual pasta do seu computador está a imagem do boneco e a posição onde ela vai ficar na tela que o **XNA** vai cuidar do resto, independentemente de o seu computador ser mais lento ou mais rápido ou de quanta memória ele tem. Moleza! Você verá esse procedimento com mais detalhes na Aula 6.

IDEALIZAÇÃO DO JOGO

Agora que você já aprendeu um pouco sobre as ferramentas para desenvolver os jogos, vamos conversar sobre a idéia do jogo em si.

Para iniciar o desenvolvimento de um jogo, antes de tudo é necessário ter uma boa idéia. Você se lembra do que vimos na aula anterior, que falava sobre as categorias de aprendizado de um jogo?

Idealizar um jogo é semelhante a criar uma história ou uma redação. Primeiro, pense em como deve ser o jogo, e que tipo de experiência ou aprendizado ele deve trazer para o seu jogador. Para idealizar um jogo, procure identificar:

- De qual ou quais categorias de aprendizado ele fará parte: Arcade, tiro, simulação espacial, construção, tradicional.
- Quais são os personagens principais do jogo? Tente visualizá-los.
- A história em torno dos personagens. Em que época o jogo vai se passar? Será um jogo futurista, medieval ou nos tempos de hoje?

É importante também estabelecer se o jogo será um novo jogo ou uma versão modificada (mod) de um jogo já existente.

Exemplo de um plano de jogo resumido:

Nome do jogo: A Vingança de Anakam: 1 – o ás do espaço.

Categoria de aprendizado: simulação (espacial).

Época: futurista.

Personagens: Anakam Spacewalker (jogador), Zark MoonVader (inimigo principal) e soldados do espaço (inimigos do jogador no espaço).

Cenário do jogo: espaço sideral. Ao fundo, diversas estrelas e corpos estelares. Ocasionalmente aparecerão planetas, conforme a movimentação da espaçonave do jogador.

História do jogo: Anakam, ao retornar de suas férias no planeta Kibokan, percebe que sua terra natal foi invadida e devastada por um ataque muito poderoso. Após conversar com alguns sobreviventes do ataque, percebe que tudo foi obra do malicioso e já conhecido Zark MoonVader, vilão inveterado das galáxias mais próximas e com domínio localizado no anel externo da galáxia Makabro, no planeta Snistrom. Anakam decide então pedir ajuda a seu melhor amigo e ambos, após alguns dias, reconstroem a espaçonave do pai, Oki Abu, que havia sido ferido mortalmente tentando defender o planeta natal. Agora, equipado com a espaçonave do pai, a poderosa B-Wang z232, Anakam e seu amigo rumam em direção às terras do poderoso vilão para vingar a morte do pai e a destruição de seu planeta.

Atividade de reflexão 1

Vamos idealizar um jogo! Seguindo as três diretrizes listadas e o exemplo do plano de jogo, enquadre seu jogo em uma ou mais categorias de aprendizado, detalhe os personagens principais e crie uma trama em torno deles.

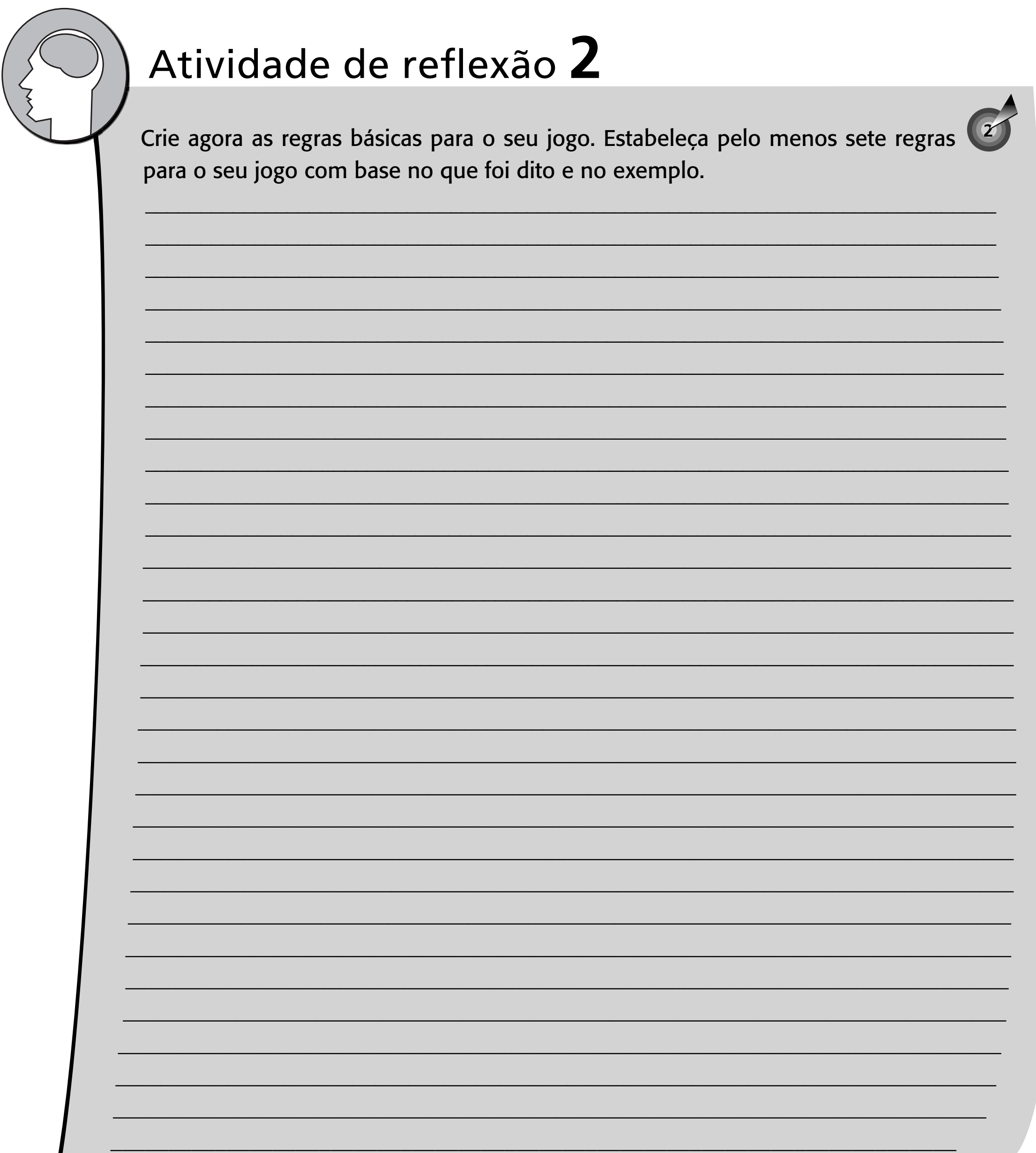
Agora que você já definiu a história e os personagens do jogo, precisa definir as regras. Por exemplo, se o jogo for de luta ou espacial, quantos pontos de vida o jogador terá ao iniciar o jogo? Quais vão ser os golpes ou armas possíveis e o dano causado por eles? Se for um jogo de tiro, quantos pontos de vida o inimigo irá perder quando for atingido por uma bala?

Exemplo de definição de regras para o jogo A Vingança de Anakam

A espaçonave do jogador (Anakam) possuirá 50 pontos de vida iniciais e uma velocidade de 5 nós estelares. A potência do tiro dessa espaçonave será de 10 megatrons. A velocidade do tiro da espaçonave será de 5 nós estelares.

As espaçonaves dos soldados do espaço terão 10 pontos de vida, uma velocidade de 3 nós estelares e uma potência de tiro de 3 megatrons. A velocidade do tiro das espaçonaves será de 2 nós estelares, ou de 4 nós estelares para os soldados experientes.

Ao ser atingida, a espaçonave do jogador perderá os pontos de vida correspondentes à potência do tiro da espaçonave inimiga. Se o total de pontos de vida atingir o valor de 0 (zero) ponto ou menos, a espaçonave explodirá, e o jogador perderá uma vida. O jogador possui um total de 3 vidas durante cada partida do jogo.



Atividade de reflexão 2

Crie agora as regras básicas para o seu jogo. Estabeleça pelo menos sete regras para o seu jogo com base no que foi dito e no exemplo.

Isso tudo que você acabou de escrever é muito importante e chama-se plano de desenvolvimento do jogo. Quando esse plano é bem feito, pode reduzir as inconsistências e erros no jogo e facilitar sua produção.

Uma vez pronto, o plano de desenvolvimento normalmente é repassado a um *designer* gráfico para que ele comece a produzir os personagens do jogo, baseando-se nos detalhes descritos. A partir desse ponto, a trilha sonora do jogo também pode começar a ser produzida, com base nas informações descritas de época, atmosfera e gênero do jogo.

Equipe de desenvolvimento do jogo

Agora que você já criou o plano de jogo, conheça mais sobre os principais profissionais envolvidos na sua produção. Normalmente, quando o jogo é muito grande ou complicado, é contratada uma equipe de desenvolvedores com pelo menos um profissional responsável por cada aspecto específico. Os principais profissionais de desenvolvimento são:

Game designer – também conhecido como roteirista, é considerado a viga mestra para o desenvolvimento do jogo. Esse profissional é responsável por elaborar a idéia e o plano de desenvolvimento do jogo, ou seja, descrever com detalhes cada aspecto do jogo, incluindo personagens, cenários e a mecânica, de forma que a idéia possa ser repassada para os outros profissionais trabalharem em seguida. Ele estará sempre interagindo com os demais profissionais, de forma a reforçar ou modificar a idéia e o roteiro do jogo. Nas Atividades 1 e 2 você assumiu o papel de um *game designer*!

Artista gráfico – também chamado de *designer* gráfico, é responsável por dar expressão e vida aos personagens e cenários que foram descritos pelo *game designer*, desenhando os modelos dos personagens (em papel e depois no computador) e os cenários, de acordo com as características definidas.

Músico – profissional responsável pela criação da trilha sonora e de todos os demais efeitos sonoros (golpes, ruídos...) existentes no jogo. Importantíssimo para dar o clima e a sensação que só bons arranjos ou composições podem trazer.

Programador – profissional responsável por unificar e concretizar tudo que já foi idealizado, planejado, desenhado e elaborado, ou seja, juntar os trabalhos dos outros profissionais e produzir um produto final, que é o jogo propriamente dito. É o responsável por fazer a coisa acontecer.

Atividade de reflexão 3

De quem você precisa para produzir o seu jogo? Defina quais dos profissionais citados você precisa para desenvolver seu jogo e quantas pessoas você acha que seriam necessárias para construí-lo. Diga também com qual ou quais profissionais você se identifica mais, ou seja, qual das funções você acharia mais interessante assumir se fizesse parte de uma equipe de desenvolvimento.

____

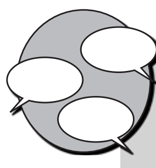
____

____

____

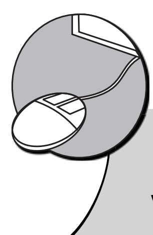
____

____



INFORMAÇÕES SOBRE FÓRUM

Qual foi o jogo que você idealizou? Entre no fórum da semana e troque informações com os amigos.



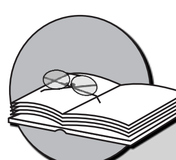
Atividade *online*

Agora que você já sabe mais sobre como idealizar um jogo, então vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- A programação de jogos pode ser bastante facilitada com o uso das ferramentas **XNA** e **Visual Studio**.
- O **Visual Studio** é uma espécie de "ajudante" que corrige os erros cometidos durante a criação do jogo.
- O **XNA** é o "amigo" que, junto com o **Visual Studio**, organiza o jogo em partes e facilita a comunicação do jogo com as outras partes do computador, como o vídeo e o som.
- Para idealizar um jogo, precisamos pensar em 3 questões:
    - o De que categoria de aprendizado ele fará parte (Arcade, tiro, simulação espacial, construção, tradicional etc.).
    - o Os personagens principais do jogo.
    - o A história em torno dos personagens e a época em que o jogo vai se passar.
- Os principais profissionais envolvidos na criação de um jogo são o *game designer*, o artista gráfico, o músico e o programador:
    - o O *game designer* é o responsável pela criação do plano de desenvolvimento do jogo, ou seja, um documento especificando todos os detalhes sobre o jogo, a história, os personagens e sua mecânica de funcionamento.

- o O artista gráfico desenha e dá vida aos personagens.
- o músico cria a trilha sonora e os efeitos do jogo.
- o O programador junta todas as partes, criando efetivamente o jogo.



Informação sobre a próxima aula

Na próxima aula, você verá como instalar as ferramentas que utilizaremos para desenvolver os nossos jogos: o **Visual Studio** e o **XNA** Game Studio. Até Lá!

AULA

Instalando o Visual C# e o XNA Game Studio

Meta da aula

Instalar corretamente as ferramentas necessárias para a criação dos jogos.

objetivos

Ao final desta aula, você deverá ser capaz de:

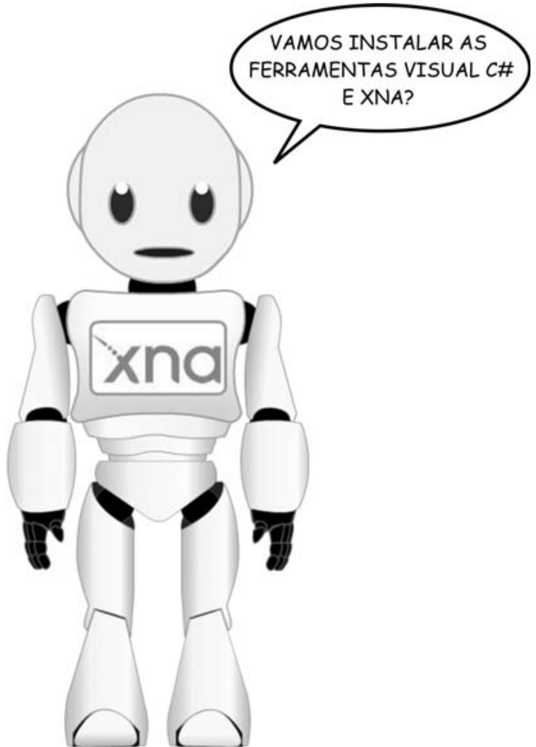
atualizar o sistema operacional Windows com os pacotes de atualização (*service packs*);

realizar a instalação das ferramentas Visual C# e XNA Game Studio.

Pré-requisitos

Conhecer os conceitos Visual Studio e XNA Game Studio, abordados na Aula 2; possuir um computador com o sistema operacional **Windows XP** ou o **Vista**.

INTRODUÇÃO



Para você iniciar a criação e a programação dos seus jogos, precisa aprender como instalar as seguintes ferramentas no seu computador:

- Visual C#.
- XNA Game Studio.

Antes de iniciar a instalação das ferramentas, verifique a versão do seu sistema operacional e qual pacote de atualizações está instalado no seu computador. Para isso, siga os três passos:

1) Acesse o painel de controle do Windows: clicando no botão "Iniciar", localizado no canto inferior direito, selecionando a opção "Configurações" e, em seguida, "Painel de Controle".

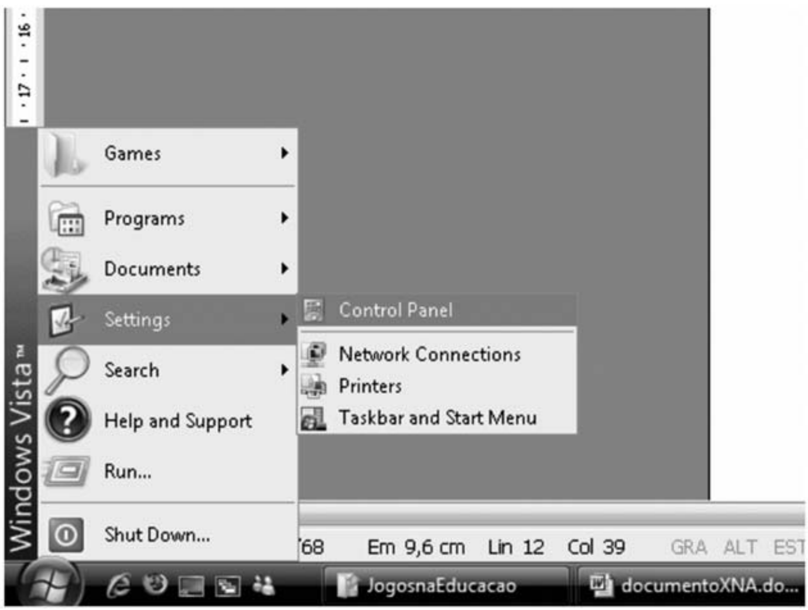


Figura 3.1: Acessando o painel de controle do Windows.
Fonte: Windows Vista.

2) Clique no ícone "Sistema": com o painel de controle aberto, procure o ícone "Sistema" dentre os exibidos no painel de controle e clique nele.

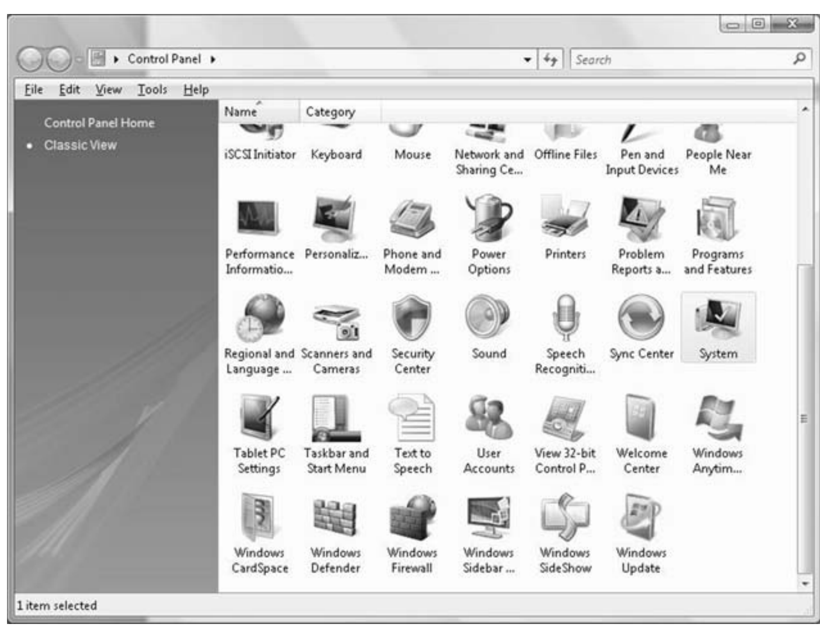


Figura 3.2: Selecionando o ícone Sistema no painel de controle.
Fonte: Windows Vista.

3) Verifique, dentro da tela “Sistema”, qual a versão do sistema operacional e o pacote de atualizações instalados no seu computador. Na figura a seguir, você poderá verificar que o sistema operacional instalado é o Windows Vista Business. Também aparece escrito “*Service Pack 1*”, indicando que o pacote de atualizações “*service pack 1*” já está instalado.

Figura 3.3: Verificando informações do Sistema Operacional.
Fonte: Windows Vista.

Atividade prática 1

Siga os três passos apresentados e verifique qual sistema operacional você possui e qual pacote de atualizações você tem instalado no seu computador.

____

____

INSTALAÇÃO DO PACOTE DE ATUALIZAÇÕES

Agora que você já sabe qual versão do sistema operacional está no seu computador e qual pacote de atualizações está instalado, verifique se seu computador está pronto para instalar as ferramentas para a produção dos jogos.

Você possui o computador com o sistema operacional **Windows XP com *service pack 3*** instalado ou então o **Windows Vista com o *service pack 1*** instalado?

Caso a sua resposta seja sim, pode pular para o próximo tópico: **instalação da ferramenta Visual C#**.

Caso contrário, aí vão os endereços de internet para você baixar o pacote de atualizações *service pack* 3 para o Windows XP e o *service pack* 1 para o Windows Vista.

Service pack 3 para o Windows XP:

- http://www.microsoft.com/downloads/details.aspx?FamilyID=5b33b5a8-5e76-401f-be08-1e1555d4f3d4&DisplayLang=pt-br

Service pack 1 para o Windows Vista:

- http://www.microsoft.com/downloads/details.aspx?FamilyID=f559842a-9c9b-4579-b64a-09146a0ba746&DisplayLang=pt-br

> Caso seu sistema operacional seja de 64 *bits*, acesse o *link* anterior e utilize a interface de busca da Microsoft para procurar o pacote de atualizações correto.

Após acessar um desses endereços, você verá uma tela semelhante à exibida a seguir. Clique no botão "Fazer *Download*" para baixar o pacote de atualizações.

Figura 3.4: Baixando o pacote de atualizações para o Windows.
Fonte: http://www.microsoft.com/downloads/details.aspx?FamilyID=5b33b5a8-5e76-401f-be08-1e1555d4f3d4&DisplayLang=pt-br

Após baixar o arquivo do pacote de atualizações, basta clicar duas vezes sobre ele para executar a atualização do seu sistema operacional. É recomendado que você feche todas as janelas abertas antes de iniciar essa operação. Após a instalação do pacote de atualizações, seu computador será reiniciado.

Atividade prática 2

Siga essas orientações instale o pacote de atualizações no seu computador. Caso encontre algum erro, escreva o erro encontrado e envie o problema para o fórum, a fim de que seus amigos possam ajudá-lo.

____

____

____

____

Volte ao início do tópico Introdução para refazer a verificação do pacote de atualizações instalado.

Instalação da ferramenta Visual C#

Após a instalação do pacote de atualizações, seu computador ficou preparado para receber a instalação das ferramentas para a criação dos jogos. Verifique então como é realizada a instalação da ferramenta Visual C#.

Acesse o *link* a seguir para realizar o *download* do Visual C#:

• http://www.microsoft.com/express/download/default.aspx

Você verá uma tela contendo vários aplicativos para *download*. Procure um retângulo verde na tela, onde está escrito "Visual C#", e clique no link "*download*" que aparece dentro do retângulo, para baixar o arquivo de instalação da ferramenta.

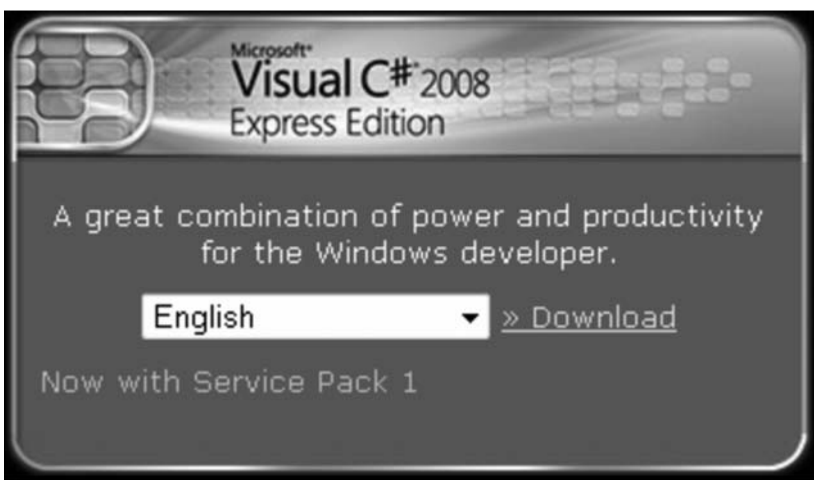


Figura 3.5: Baixando a ferramenta Visual C#.
Fonte: http://www.microsoft.com/express/download/default.aspx

Após baixar o arquivo de instalação da ferramenta Visual C#, será necessário clicar nesse arquivo duas vezes para iniciar a instalação. Repare que, durante a instalação, ele continuará baixando os demais componentes da ferramenta pela internet.

Atividade prática 3

Siga as orientações e instale a ferramenta Visual C# no seu computador. Caso encontre algum erro, escreva o erro encontrado e envie o problema para o fórum, a fim de que seus amigos possam ajudá-lo.

____

____

____

____

Instalação da ferramenta XNA Game Studio

Você já está quase chegando lá! Só falta instalar a última ferramenta – e a mais importante: o XNA Game Studio.

Para isso, acesse o endereço a seguir para baixar o arquivo de instalação dessa ferramenta:

- http://www.microsoft.com/downloads/details.aspx?FamilyID=7d70d6ed-1edd-4852-9883-9a33c0ad8fee&DisplayLang=en

Após acessar, você verá uma tela semelhante à tela a seguir. Clique no botão "*Download*" para baixar o arquivo de instalação do XNA Game Studio.

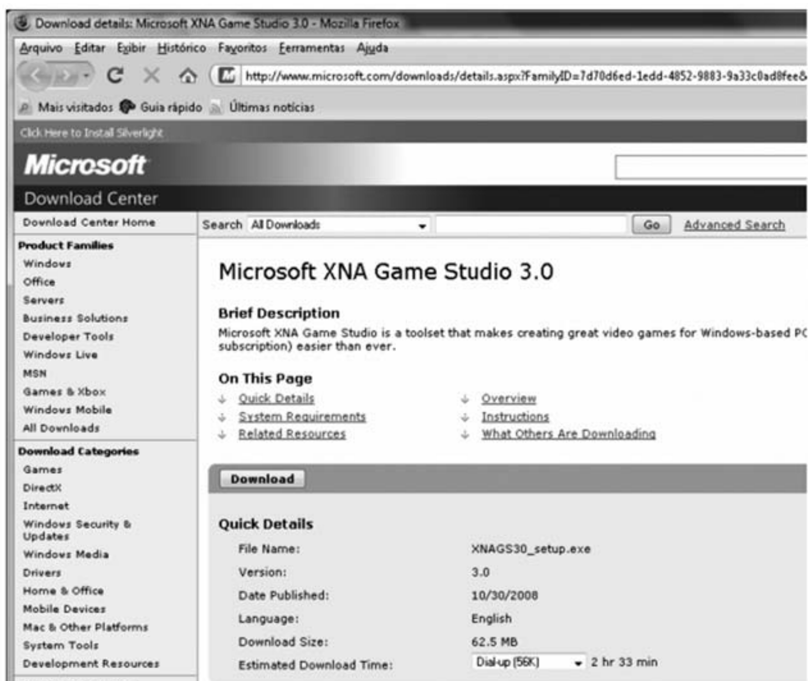
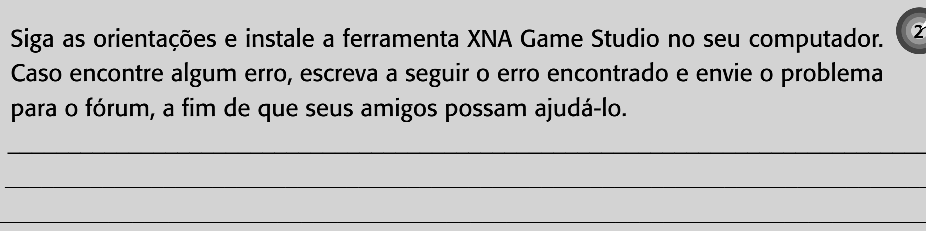


Figura 3.6: Baixando a ferramenta XNA Game Studio.
Fonte: http://www.microsoft.com/downloads/details.aspx?FamilyID=7d70d6ed-1edd-4852-9883-9a33c0ad8fee&DisplayLang=en

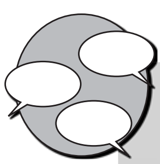
Após baixar o arquivo, será necessário clicar nele duas vezes para iniciar a instalação.

Atividade prática 4



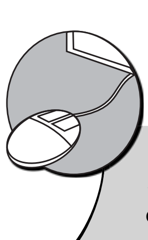
Siga as orientações e instale a ferramenta XNA Game Studio no seu computador. Caso encontre algum erro, escreva a seguir o erro encontrado e envie o problema para o fórum, a fim de que seus amigos possam ajudá-lo.

Parabéns! Agora você já está com o seu computador preparado para iniciar o desenvolvimento de jogos utilizando a tecnologia XNA, da Microsoft.



INFORMAÇÕES SOBRE FÓRUM

Você teve alguma dificuldade para instalar o pacote de atualizações ou as ferramentas? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.

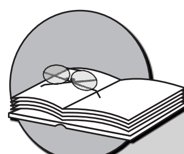


Atividade *online*

Agora que você já está com o computador pronto para desenvolver seus jogos, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- Para você iniciar a criação e a programação dos seus jogos, precisa aprender como instalar as ferramentas Visual C# e XNA Game Studio no seu computador.
- O primeiro passo é verificar se seu computador possui instalado o sistema operacional **Windows XP com *service pack 3*** ou o **Windows Vista com o *service pack 1***. Caso contrário, não será possível instalar as ferramentas para o desenvolvimento dos jogos.
- Os endereços para baixar os arquivos de instalação das ferramentas Visual C# e XNA Game Studio são:
    - http://www.microsoft.com/express/download/default.aspx
    - http://www.microsoft.com/downloads/details.aspx?FamilyID=7d70d6ed-1edd-4852-9883-9a33c0ad8fee&DisplayLang=en



Informação sobre a próxima aula

Na próxima aula, você verá como criar e codificar classes de objetos utilizando a linguagem Visual C#.

Criando a classe Espaçonave

Meta da aula

Mostrar como funciona a programação orientada a objeto.

objetivos

Ao final desta aula, você deverá ser capaz de:

1. conceituar atributo, método, classe e objeto;
2. criar e codificar em C# uma classe para seu jogo, com métodos e atributos próprios.

Pré-requisitos

Conhecer a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um plano de desenvolvimento do jogo, construído na Aula 2.

INTRODUÇÃO

Na Aula 2, idealizamos um jogo e geramos seu plano de desenvolvimento, um passo essencial para evitar problemas na hora de escrever o código desse jogo.

O.k.! Vamos rever o plano de desenvolvimento do jogo produzido na Aula 2.

Nome do jogo: A vingança de Anakam: 1 – o ás do espaço.

Categoria de aprendizado: simulação (espacial).

Época: futurista.

Personagens: Anakam Spacewalker (jogador), Zark MoonVader (inimigo principal) e soldados do espaço (inimigos do jogador no espaço).

Cenário do jogo: espaço sideral. Ao fundo, diversas estrelas e corpos estelares. Ocasionalmente aparecerão planetas, conforme a movimentação da espaçonave do jogador.

História do jogo: Anakam, ao retornar de suas férias no planeta Kibokan, percebe que sua terra natal foi invadida e devastada por um ataque muito poderoso. Após conversar com alguns sobreviventes ao ataque, percebe que tudo foi obra do malicioso e já conhecido Zark MoonVader, vilão inveterado das galáxias mais próximas, que tem seu domínio localizado no anel externo da galáxia Makabro, no planeta Snistrom. Anakam decide então pedir ajuda a seu melhor amigo e ambos, após alguns dias, reconstroem a espaçonave do pai Oki Abu, que havia sido ferido mortalmente tentando defender o seu planeta natal. Agora, equipado com a espaçonave do pai, a poderosa B-Wang z232, Anakam e seu amigo rumam em direção às terras do poderoso vilão para vingar a morte do pai e a destruição de seu planeta.

Regras do jogo: a espaçonave do jogador (Anakam) possuirá 50 pontos de vida iniciais e velocidade de 5 nós estelares. A potência de tiro dessa espaçonave será de 10 megatrons. A velocidade do tiro da espaçonave será de 5 nós estelares.

As espaçonaves dos soldados do espaço terão 10 pontos de vida, velocidade de 3 nós estelares e potência de tiro de 3 megatrons. A velocidade do tiro das espaçonaves será de 2 nós estelares, ou de 4 nós estelares para os soldados experientes.

Ao ser atingida, a espaçonave do jogador perderá os pontos de vida correspondentes à potência do tiro da espaçonave inimiga. Se o total de pontos de vida atingir o valor de 0 (zero) ponto ou menos, a espaçonave explodirá, e o jogador perderá uma vida. O jogador possui 3 vidas durante cada partida do jogo.

E agora, como você fará para começar a escrever o código desse jogo?

Repare que, nas regras do nosso plano de jogo espacial, podemos verificar que a ideia de uma nave aparece duas vezes, uma para a espaçonave do jogador e outra para as naves inimigas dos soldados do espaço.

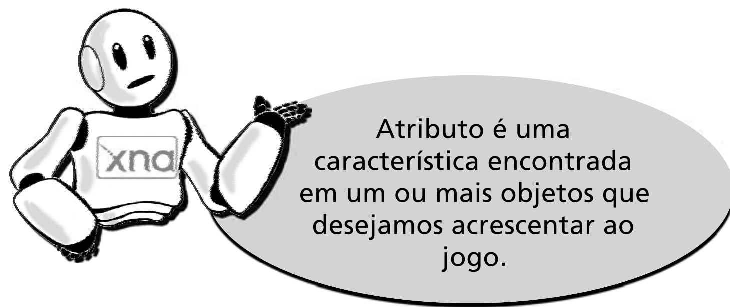
Figura 4.1: Exemplos de espaçonaves.
Fonte: Cooliris.

Agora imagine como seriam essas espaçonaves. Ao comparar as duas espaçonaves, você precisa responder a algumas perguntas:

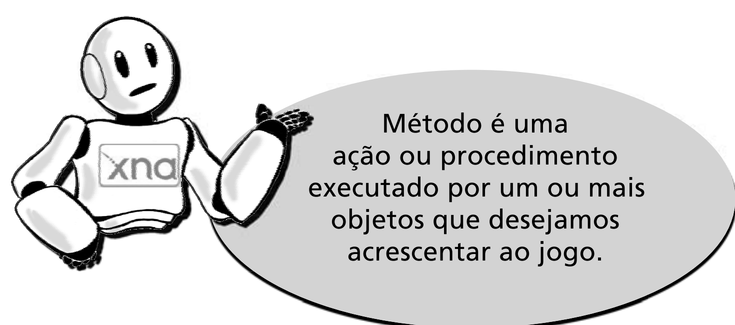
- A espaçonave do jogador é mais forte ou mais fraca que a dos soldados do espaço?
- A espaçonave do jogador é mais rápida do que a dos soldados do espaço?
- A espaçonave do jogador solta um tiro diferente daquele dos soldados do espaço?
- A espaçonave do jogador e a dos soldados do espaço são totalmente diferentes ou têm algo em comum?

Veja que, ao responder às três primeiras perguntas, você está "configurando" diferentes espaçonaves. A primeira pergunta fala sobre vida, a segunda fala sobre velocidade e a terceira, sobre potência do tiro da espaçonave. Observe

que ambas as espaçonaves possuem estas três características: vida, velocidade e potência do tiro. A essas características damos o nome de atributos.



Perceba também que, além dos atributos, as espaçonaves executarão ações durante o jogo. Pense em quais ações as espaçonaves poderiam executar: mover, atirar e levantar escudo são alguns exemplos. A essas ações executadas por ambas as espaçonaves damos o nome de métodos.

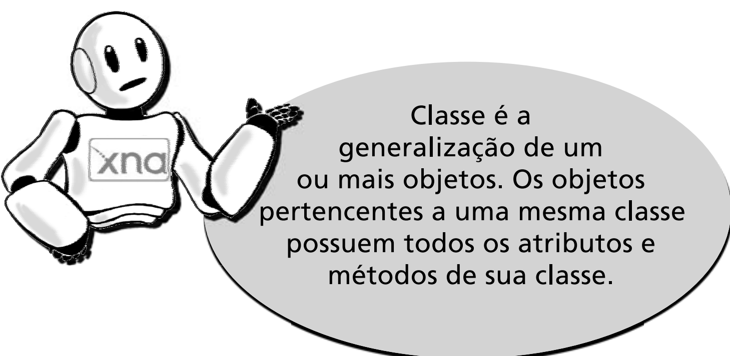


Objeto árvore

Classe
Árvore
??????
(Qualquer uma.
Abstrata.)

Figura 4.2: Conceito de classe.
Fonte: http://www.linhadecodigo.com.br/artigos/img_artigos/robert_martim/image007.png

Veja que a quarta pergunta já foi respondida também. Ambas as espaçonaves possuem três atributos em comum: vida, velocidade e potência do tiro. Os métodos Mover, Atirar e Levantar Escudo também são comuns a ambas. Quando isso acontece, podemos dizer que ambas as espaçonaves pertencem a uma mesma classe de objetos.



Você pode, então, criar uma classe de objetos chamada Espaçonave, que possui os atributos vida, velocidade e potência do tiro e os métodos Atirar, Mover e Levantar Escudo. A nave do jogador e as naves inimigas dos soldados do espaço são objetos que podem pertencer à classe de objetos Espaçonave.

O conceito de orientação a objeto é muito antigo, teve suas origens na década de 1960, quando se começou a discutir o problema do desenvolvimento de aplicativos grandes mantendo um bom padrão de qualidade. A partir de então, começaram a surgir algumas linguagens que continham em si os conceitos de orientação a objeto. A linguagem C é uma delas.

Atividade 1

Com base no plano de jogo que foi elaborado na Aula 2, escolha um dos personagens e defina uma classe para ele, composta de pelo menos 5 atributos e 3 métodos. Escreva a seguir o nome da classe, os atributos e os métodos.

____
____
____
____
____
____
____
____
____
____

CODIFICANDO A CLASSE ESPAÇONAVE EM C#

Pois bem, uma vez que já definimos os atributos e métodos da nossa classe Espaçonave, escreva-a utilizando a linguagem C#. Veja como você pode escrever o código da classe Espaçonave em C#:

```
    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
        }
    }
```

Repare que a primeira palavra que aparece no código é **public**. O termo **public**, em português, significa público. Para o Visual C#, isso também indica que a classe Espaçonave é de domínio público, ou seja, estará disponível e poderá ser acessada em qualquer parte do código do jogo ou ainda por outro programa.

Repare que, na linha abaixo, após o comando "public class Espaconave", existe a chave "{". As chaves "{" e "}" indicam o início e o fim de um trecho de código. Nesse caso, estão indicando para o Visual C# onde começa e onde termina a nossa classe Espaçonave.

Preste atenção neste trecho:

```
private string _nome;
private int _energia;
private int _velocidade;
private int _potencia_tiro;
private int _velocidade_tiro;
private int _escudo;
```

Está reconhecendo alguns nomes de atributos que criamos? Exatamente! Esse trecho define todos os atributos da classe Espaçonave. O termo **private**, ou privado, sinaliza para o Visual C# que todos os atributos da classe Espaçonave serão privados, ou seja, apenas os

métodos criados dentro da própria classe Espaçonave poderão acessar esses atributos.

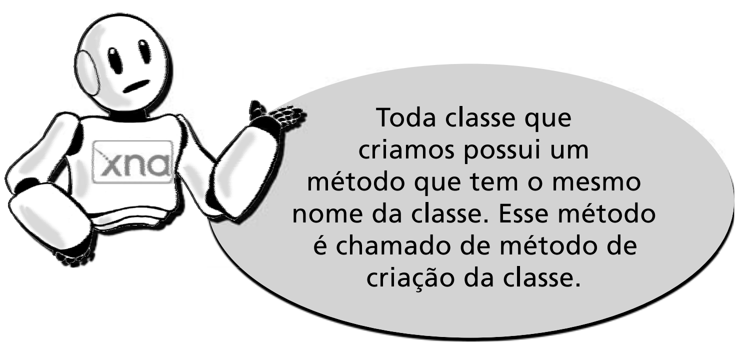
Perceba que as palavras **string** e **int** aparecem antes do nome dos atributos. Elas servem para indicar de que tipo são os atributos. O atributo **_nome**, por exemplo, é do tipo **string**, ou seja, um conjunto de caracteres para guardarmos o nome da espaçonave. Já o atributo **_energia** é do tipo **int**, ou seja, só pode armazenar números inteiros. Esse atributo vai dizer quantos pontos de vida a espaçonave ainda tem.

Agora observe este outro trecho:

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

A princípio, parece um tanto estranho colocar o nome Espaçonave novamente. Mas isso tem uma explicação.

Dentro do método de criação da classe, colocamos valores iniciais



para nossos atributos mais importantes. Veja que os atributos energia, velocidade e velocidade do tiro receberam valores iniciais. Isso mostra que toda espaçonave que for criada dentro do jogo inicialmente possuirá 50 pontos de vida, 5 pontos de velocidade e 5 pontos na velocidade do tiro. É claro que esses valores poderão ser alterados mais tarde.

Por fim, veja o último trecho do código da nossa classe Espaçonave:

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

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

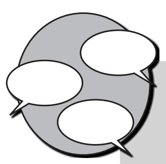
Está reconhecendo esses nomes? Sim! São os métodos **Mover** e **Atirar**, que criamos para a nossa classe Espaçonave. Logo após o nome **Mover** existe um texto dentro dos parênteses, que é "**int direcao**". Para o Visual C#, isso quer dizer que, para movermos a espaçonave, precisamos fornecer uma direção de destino, ou seja, um número inteiro (**int**) que indica em qual direção a espaçonave irá se mover quando esse método for chamado. Chamamos esses valores de **parâmetros de entrada** do método Mover.

Existe também uma palavra estranha que vem antes do nome do método, que é o **void**. Em português, **void** significa "vácuo"! Isso quer dizer que o método Mover, ao ser chamado, não retornará um vácuo, ou seja, nenhum valor para quem o chamou. Apenas fará a movimentação da espaçonave.

Dentro dos métodos Mover e Atirar, existem textos na cor verde e com duas barras "//" antes. Quando colocamos essas duas barras antes do texto, mostramos para o Visual C# que o que estamos escrevendo são apenas comentários dentro do código da classe. Isso indica também que os métodos **Mover** e **Atirar** ainda não foram efetivamente codificados...

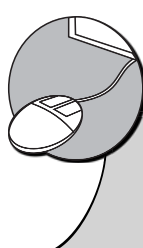
Atividade 2

Com base no exemplo de código apresentado, escreva na linguagem Visual C# o código para descrever a classe que você idealizou na Atividade 1 desta aula.



INFORMAÇÕES SOBRE FÓRUM

Você agora reconhece como funcionam as classes, os atributos e os métodos? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.



Atividade *online*

Agora que você idealizou a sua classe de objetos, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- Para transferir para o Visual C# a ideia do seu jogo, é importante compreender o conceito de classe. Imaginando um jogo composto pelas espaçonaves do jogador e inimigas, você pode generalizar o conceito de espaçonave e, com base nele, definir as características e as ações que toda espaçonave no seu jogo deve possuir.
- As características são os atributos.
- As ações possíveis são os métodos.
- Atributos e métodos, como um todo, compõem uma classe.
- A fim de representar as espaçonaves do jogo, podemos criar então uma classe chamada Espaçonave, com os atributos vida, velocidade e velocidade do tiro e os métodos Mover, Atirar e Levantar Escudo.
- Toda classe possui um método principal, que possui o mesmo nome da classe. Esse método é o método de criação da classe, e nele são definidos os valores iniciais para os atributos principais da classe.
- As palavras **public** e **private** indicam se os atributos ou métodos são públicos ou privados, ou seja, se podem ser acessados livremente (**public**) ou apenas dentro do código da própria classe onde foram criados (**private**).

Informação sobre a próxima aula

Na próxima aula, você verá como iniciar um projeto de jogo dentro da ferramenta que acabou de instalar no seu computador, o Visual C#!

Iniciando o projeto de um jogo no Visual C#

Metas da aula

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

objetivos

Ao final desta aula, você deverá 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

Conhecer a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na Aula 3; conhecer os conceitos de classe e objeto, abordados na Aula 4.

INTRODUÇÃO

Na aula anterior, você estudou 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, 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#.
Fonte: Visual C#.

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

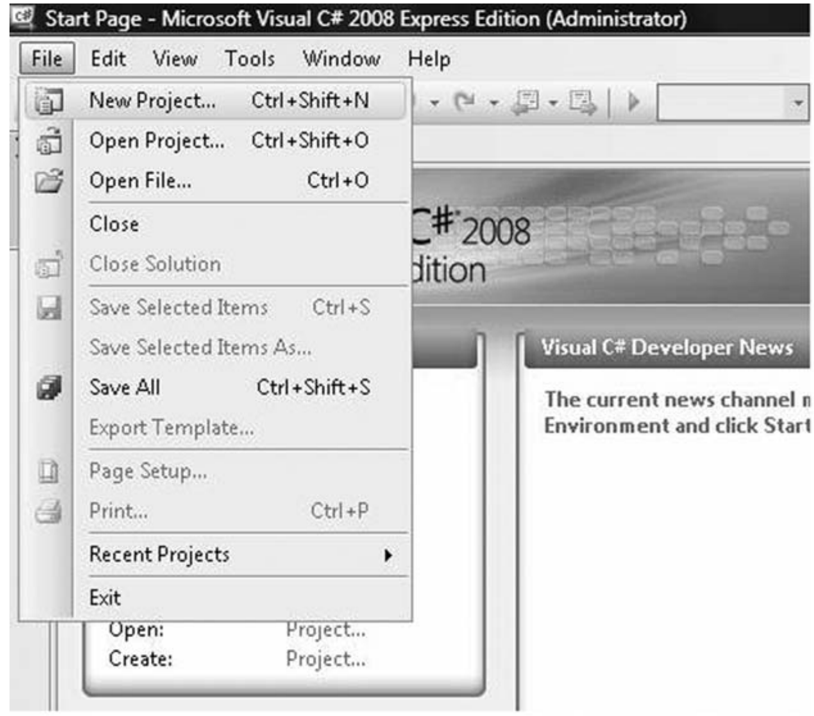


Figura 5.2: Opção File/New Project, para criar um novo projeto.
Fonte: Visual C#.

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 do 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 inferior 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.
Fonte: Visual C#.

Atividade prática 1

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

ENTENDENDO A ESTRUTURA DE UM PROJETO DE JOGO

Após ter criado 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, você observa uma série de instruções contendo a palavra **using**. 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 no nosso código outras bibliotecas de comandos.

Por exemplo, ao escrevermos

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

é como se você estivesse numa 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 seção XNA e, dentro dela, procure uma prateleira rotulada *Framework*. Nela, você encontrará os livros de áudio.”

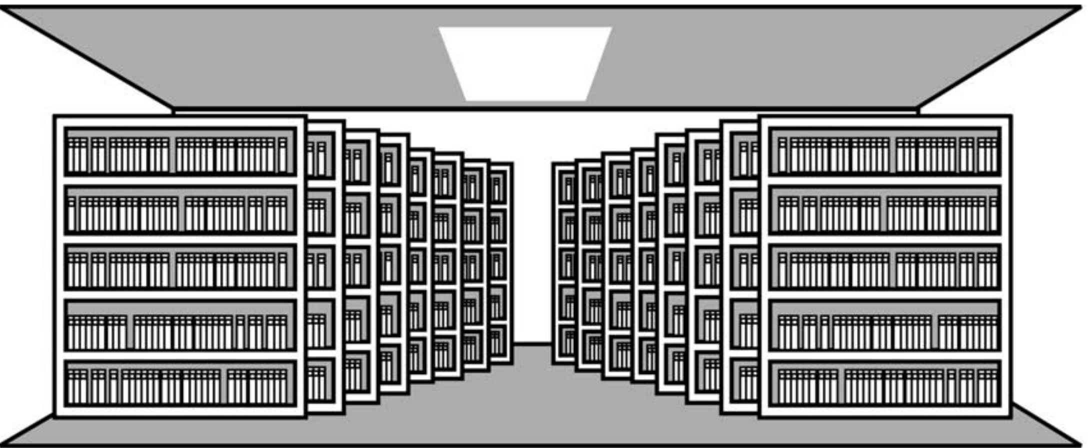
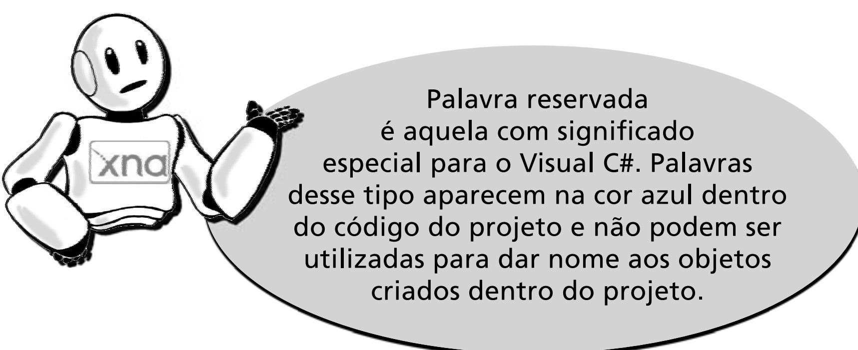


Figura 5.4: Biblioteca.

Dessa forma, ao chamar essa instrução, 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 a áudio, ou seja, comandos prontos para lidar com os sons do nosso jogo.

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

Repare também que apenas a palavra **using** aparece na cor azul. Isso significa que o Visual C# reconheceu essa palavra e sinalizou, com a cor azul, que **using** é uma palavra reservada.



Agora veja o que significa uma outra palavra reservada, chamada **namespace**:

```
namespace JogoEspacial
{
```

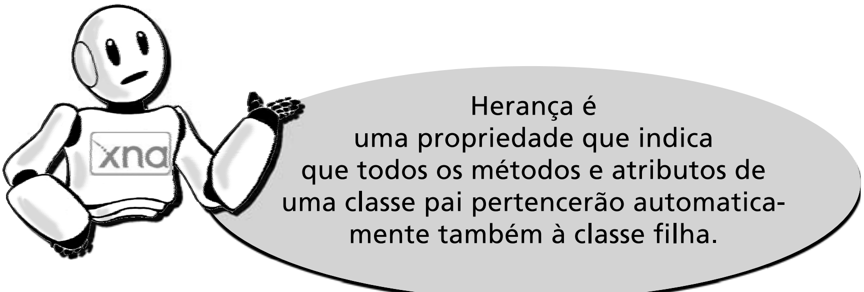
Traduzindo para o português, **namespace** significa "nome de espaço". Ao utilizar a instrução **namespace** JogoEspacial, estamos mostrando para o Visual C# que criamos uma área chamada "JogoEspacial" onde colocaremos instruções do nosso jogo. Normalmente, o **namespace** possui o mesmo nome do projeto.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

Repare que na linha seguinte ao 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 a seguir. Você se lembra do conceito de classe que aprendeu 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. Isso quer dizer que a classe Game1 é filha da classe Microsoft.Xna.Framework.Game. Dessa forma, a classe Game1 possui todos os atributos e métodos que a classe Microsoft.Xna.Framework.Game, a classe pai, possui. A essa propriedade damos o nome de "herança".



Repare agora no seguinte trecho:

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

Estes são os atributos novos da classe Game1:

- **Graphics**, do tipo GraphicDeviceManager – é o atributo que representa 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 Game1, estamos inicializando os dois atributos novos da classe. Ambos os atributos, **graphics** (vídeo) e **content** (conteúdo), são de tipo complexo e precisam da instrução **new** para serem efetivamente criados. Dentre os tipos simples, podemos listar aqueles que vimos na Aula 4: 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();
}
```

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

Esse método foi criado para você colocar valores iniciais para os objetos do seu jogo. Você poderia, por exemplo, colocar aqui 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 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**(), que serve para chamar o método Initialize da classe pai, a **Microsoft.XNA.Network.Game**. Ela foi colocada aqui para não perdermos tudo que foi codificado no método **Initialize** original, que foi herdado da classe **Microsoft.XNA.Network.Game**.

Veja 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
        }
```

Esses 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.

Veja agora o método **Update**:

```
        protected override void Update(GameTime game-
Time)
        {
            // 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);
        }
```

Esse 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 se ela foi destruída, checando os pontos de vida dela, por exemplo. Aqui você coloca as regras do jogo.

Também é utilizado para definir ações a serem realizadas quando o usuário apertar alguma tecla especial, como a “Esc”, para sair do jogo. É exatamente essa verificação que o XNA já coloca para você dentro do código, pela instrução:

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

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

Repare que esse método recebe também um parâmetro de entrada, o **gameTime**, ou seja, o tempo do jogo; esse parâmetro será 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);
        }
```

Esse 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, seja 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 desse método.

Repare que o XNA já coloca um comando dentro desse método, o **GraphicsDevice.Clear**. Esse 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 acha?

Atividade prática 2

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

____

____

____

____

____

____

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 esse código dentro do seu projeto, basta colocá-lo logo abaixo da classe Game1, após a chave "}", que indica o fim da classe.

Contudo, esse 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 Espaçonave dentro do jogo, que vai ser a nave do jogador. Vamos dar a esse objeto o nome NaveJogador. Podemos fazer isso no início do código do jogo, onde já estão definidos todos os atributos principais (classe Game1). Veja só como fica:

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 essa tarefa é dentro do método de inicialização do jogo, o **Initialize**:

```
protected override void Initialize()
 {
 // Criando efetivamente a espaçonave do joga-
 dor 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, o Visual C# sublinhou a palavra **_nome** com a cor **vermelha** no código. Veja só:

```
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";
                 string Espaconave._nome
    base.Initial
}                Error:
                   'JogoEspacial.Espaconave._nome' is inaccessible due to its protection level
```

Figura 5.5: Atributo _nome inacessível.
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. Por quê? Verifique na definição da classe Espaçonave como foi criado o atributo **_nome**:

```
private string _nome;
```

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

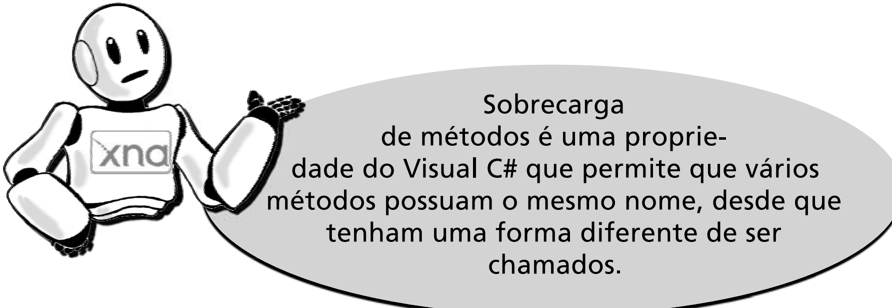
Vamos criar dois novos métodos dentro da classe Espaçonave: um para atualizar o nome da espaçonave e outro para trazer esse nome quando precisarmos. Veja como fazer isso:

```
// 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**. Ele 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. Isso 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 essa propriedade damos o nome de sobrecarga de métodos.



Agora que temos o método de atualização do nome da espaçonave, vamos corrigir o 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 prática 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.

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

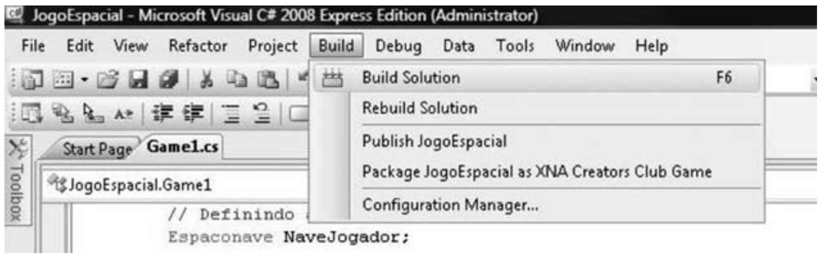


Figura 5.6: Acessando a opção Construir Solução.
Fonte Visual C#.

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

Figura 5.7: Construção de solução bem-sucedida.
Fonte Visual C#.

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

```
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    xGraphicsDevice.Clear(Color.Black);

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
```

Error List
1 Error | 0 Warnings | 0 Messages
Description
1 The name 'xGraphicsDevice' does not exist in the current context

Figura 5.8: Listagem de erros encontrados no projeto.
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 que o código seja executado.

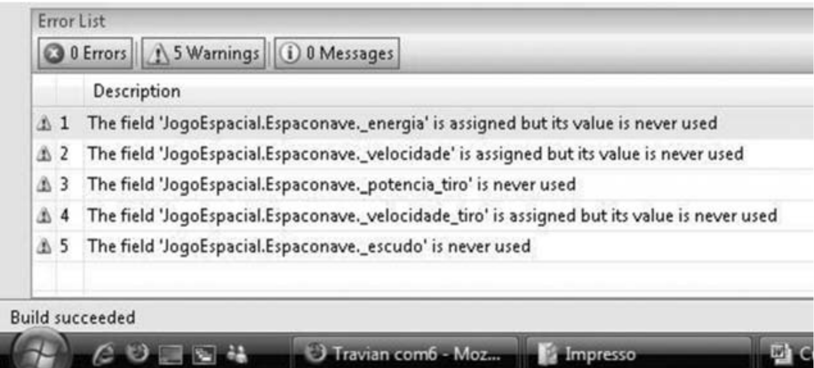
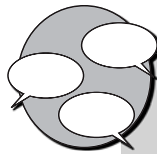


Figura 5.9: Listagem de avisos encontrados no projeto.
Fonte Visual C#.

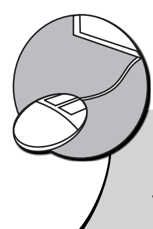
Atividade prática 4

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



INFORMAÇÕES SOBRE FÓRUM

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

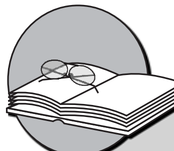


Atividade *online*

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

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: **using**, **namespace**, **public** e **class**, por exemplo. Essas 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 seu jogo.
- Para você incluir no seu projeto de jogo o código da classe Espaçonave, que foi visto na Aula 4, 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; basta que você clique duas vezes sobre os erros listados para ir até o local onde está ocorrendo o problema. Os erros impedem que o código seja executado.



Informação sobre a próxima aula

Na próxima aula, você verá como exibir e movimentar a espaçonave do jogador com o Visual C#. Até lá!

AULA

Exibindo e movimentando a sua espaçonave

Metas da aula

Exibir e movimentar a espaçonave do jogador na tela do jogo.

objetivos

Ao final desta aula, você deverá ser capaz de:

exibir a espaçonave na tela do seu projeto de jogo;

movimentar a espaçonave pela tela do seu projeto de jogo.

Pré-requisitos

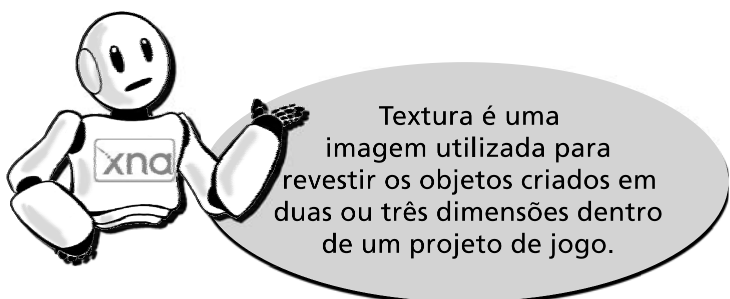
Conhecer a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na Aula 3; conhecer os conceitos de classe e objeto, abordados na Aula 4; ter iniciado seu projeto de jogo com a ferramenta Visual C# e acrescentado a classe Espaçonave ao jogo, conforme visto na Aula 5.

INTRODUÇÃO

Na aula anterior, iniciamos o nosso projeto de jogo espacial dentro da ferramenta Visual C#. Também incorporamos a ele a classe espaçonave, vista na Aula 3.

Já criamos a espaçonave do jogador. Ela possui o nome **Falcão Vingador**; os atributos de energia e velocidade também já foram configurados. Precisamos agora fazer a espaçonave aparecer na tela do jogo.

Vamos desenhar a nossa espaçonave na tela utilizando o recurso de texturas.



Veja na **Figura 6.1** alguns exemplos de imagens que podemos utilizar como texturas para as espaçonaves do jogador e as naves inimigas:

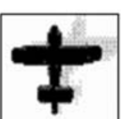
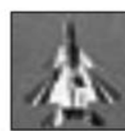
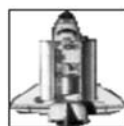
  

Figura 6.1: Imagens de aviões e espaçonaves.
Fonte: Desconhecida.

As texturas também podem ser utilizadas para dar mais realismo a outros objetos criados dentro do jogo. Na **Figura 6.2**, podemos observar alguns exemplos de texturas que podem ser usadas para revestir objetos como paredes, portas e pisos criados dentro do jogo:

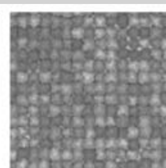


Figura 6.2: Texturas de tecido, metal, pastilhas de vidro, pedras e mármore.
Fonte: Desconhecida.

EXIBINDO A ESPAÇONAVE NA TELA DO JOGO

Reveja a nossa 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;
    }

    // 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;
    }

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

    public void Atirar()
```

```
        {
        // Cria um tiro
        }
    }
```

Passo 1: Analise agora os atributos da classe Espaçonave. Repare que ainda não temos nenhum atributo que possa armazenar uma textura para a nave. Criaremos então o atributo **_textura**.

Passo 2: Também precisamos armazenar a posição da espaçonave dentro da tela do jogo. Como estamos projetando um jogo em duas dimensões (x e y), essa posição pode ser definida por dois novos atributos: um para controlar a posição horizontal da nave (x) e outro para controlar a posição vertical (y). Podemos chamá-los de **_posicaox** e **_posicaoy**.

Após acrescentarmos os novos atributos, a classe Espaçonave ficará assim:

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

          // Novos atributos
          private int _posicaox;
          private int _posicaoy;
          private Texture2D _textura;
```

Passo 3: Observe agora o método de criação da classe Espaçonave. Tente agora definir a posição inicial da espaçonave na tela. Você pode fazer isso colocando valores para os atributos _posicaox e _posicaoy que acabou de criar. Veja como ficou o código:

```
    public Espaconave()
    {
        _energia = 50;
        _velocidade = 5;
```

```
        _velocidade_tiro = 5;
        _posicaox = 10;
        _posicaoy = 10;
    }
```

O.k. Você já criou os três novos atributos e definiu valores iniciais de posição da nave. Como faremos para carregar a imagem da espaçonave como uma textura?

Passo 4: Crie um novo método para a classe Espaçonave, chamado CarregarTextura. Veja a seguir como ficou o código:

```
// Carrega uma textura a partir de um arquivo de
imagem
public void CarregarTextura(GraphicsDevice
p_dispositivo, string p_local)
  {
 _textura = Texture2D.FromFile(p_dispositivo,
p_local);
  }
```

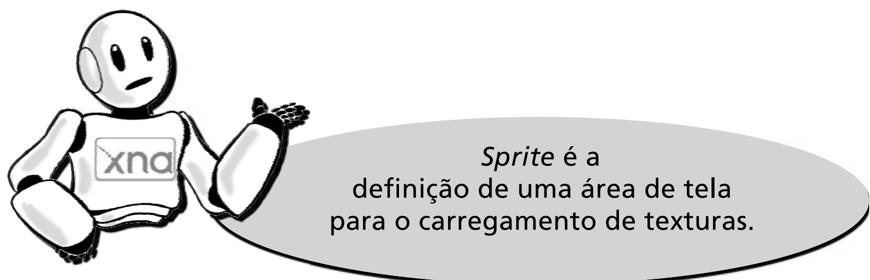
Repare que o método CarregarTextura recebe dois parâmetros: **p_dispositivo** e **p_local**. O parâmetro **p_dispositivo** é o dispositivo gráfico onde a textura será exibida, e o parâmetro **p_local** indica o caminho completo onde a imagem está localizada no computador. Se a imagem da nave do jogador estiver na pasta *imagens*, por exemplo, o caminho poderá ser "**\imagens\espaconave_jogador.jpg**".

Note também que ele chama o método **Texture2D.FromFile** para realizar a carga da textura de duas dimensões. A imagem localizada em **p_local** será carregada como textura e ficará associada ao dispositivo gráfico **p_dispositivo**, para que possa ser exibida mais tarde.

O primeiro deles é um método para carregar a textura a partir de um arquivo de imagem, chamado CarregarTextura. Repare que utilizamos o comando Texture2D.FromFile, de forma a buscar a textura a partir de um caminho de arquivo que foi informado, como por exemplo "../../../Content/Imagens/nave.PNG".

Passo 5: Já temos a textura carregada e uma posição inicial (x e y) onde a textura aparecerá na tela. Vamos criar agora o método **Desenhar** da classe Espaçonave para desenhar a textura na tela do computador, na posição (x e y) especificada.

Para que possamos desenhar a textura da espaçonave na tela, precisamos delinear uma área na tela que será ocupada por essa textura. Esta área é chamada de **sprite**.



Veja como ficou nosso método desenhar:

```
      // Desenha a nave na tela utilizando um
      sprite
  public void Desenhar(SpriteBatch p_sprite)
  {
      p_sprite.Begin();
      p_sprite.Draw(_textura, new Rectangle
      (_posicaox, _posicaoy, _textura.Width,
      _textura.Height), Color.White);
      p_sprite.End();
  }
```

Esse método recebe o parâmetro de entrada **p_sprite**, que é do tipo **SpriteBatch**. **SpriteBatch** significa "conjunto de *sprites*". Dentro desses *sprites*, colocaremos todas as texturas que serão exibidas dentro do nosso jogo espacial.

Os métodos **p_sprite.Begin**() e **p_sprite.End**() iniciam e finalizam o processo de desenhar a espaçonave na tela.

Analise agora o método **p_sprite.Draw**. Ele desenhará a textura na tela. Ele recebe como parâmetros a textura da espaçonave, que é

_textura e a definição de um retângulo onde ficará desenhada a textura. O retângulo será desenhado nas posições **_posicaox** e **_posicaoy** da espaçonave e terá a altura e a largura iguais às da textura que está sendo desenhada. Além disso, também é estabelecida uma cor de fundo branca (Color.White) para o desenho.

Atividade prática 1

Seguindo os cinco passos descritos, abra seu projeto de jogo espacial na ferramenta Visual C# e altere o código da classe Espaçonave dentro do seu projeto de jogo para acrescentar a ele os atributos e os métodos necessários para carregar e desenhar as texturas.

Nossa classe Espaçonave já está preparada para exibir a espaçonave do jogador na tela do jogo. Precisamos agora encontrar uma imagem para a espaçonave do jogador.

Vamos agora incluir a imagem da espaçonave dentro do projeto de jogo.

Passo 1: Dentro do *Solution Explorer*, localizado no canto direito do seu projeto de jogo, procure pelo item *Content* (Conteúdo). Em seguida, clique com o botão direito do mouse em cima desse item e selecione a opção *Add/New Folder*, para criar uma nova pasta chamada Imagens, conforme a **Figura 6.3**.

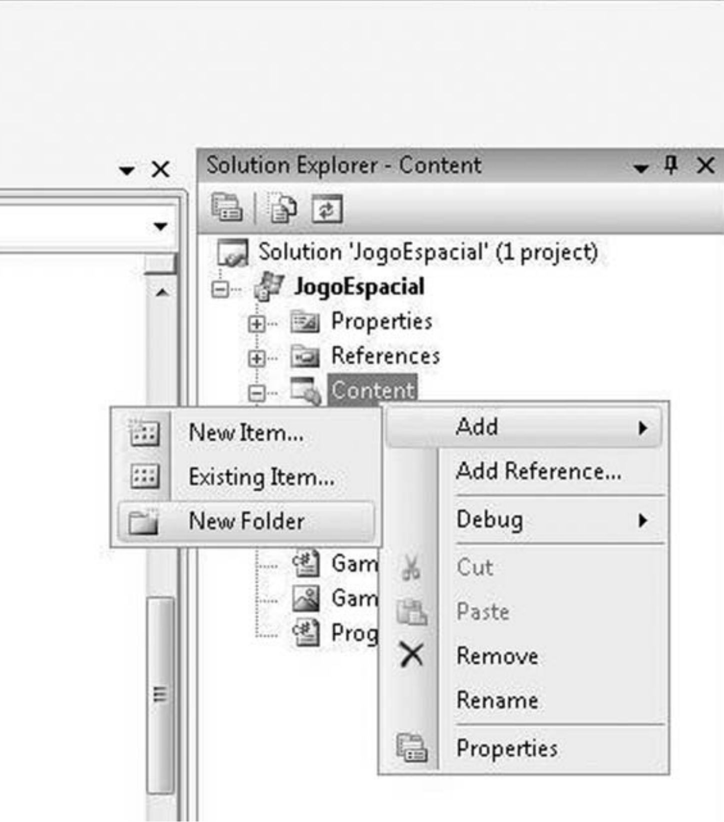


Figura 6.3: Adicionando a pasta Imagens ao seu projeto de jogo.
Fonte: Visual C#.

Passo 2: Clique com o botão direito do mouse em cima da pasta **Imagens**, que você acabou de criar, e selecione a opção *Add/Existing Item*, conforme a **Figura 6.4**.

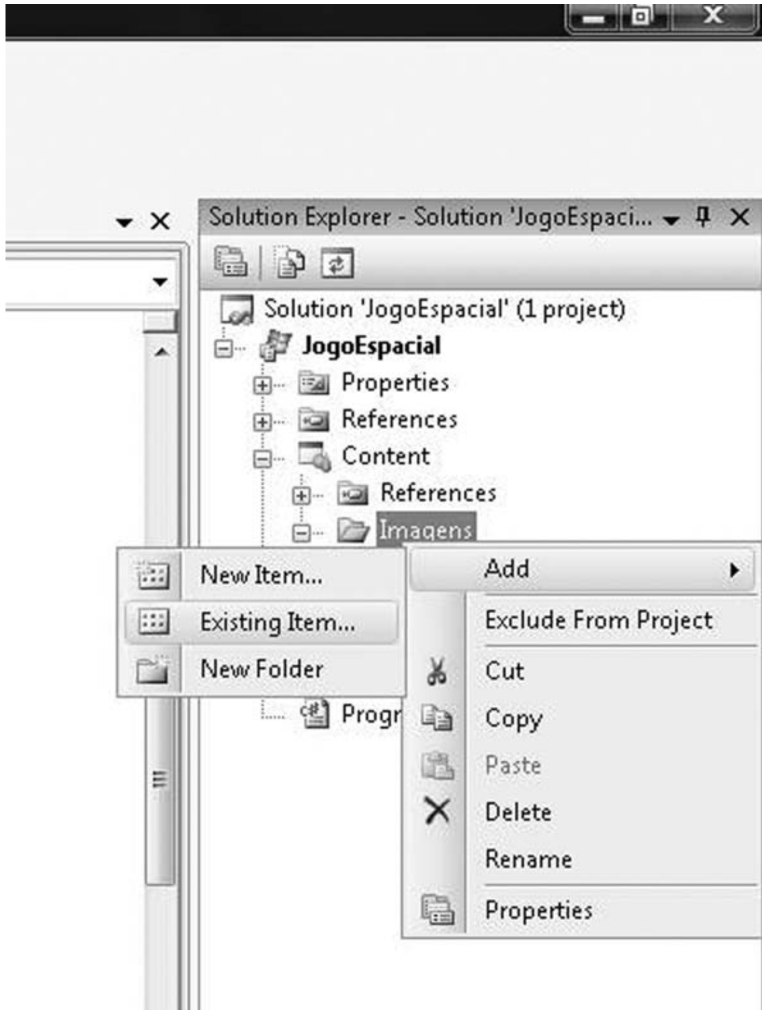


Figura 6.4: Acrescentando a imagem da espaçonave na pasta Imagens.
Fonte: Visual C#.

Passo 3: Procure o arquivo de imagem da espaçonave, selecione esse arquivo e clique no botão **Add**. Veja a **Figura 6.5**.

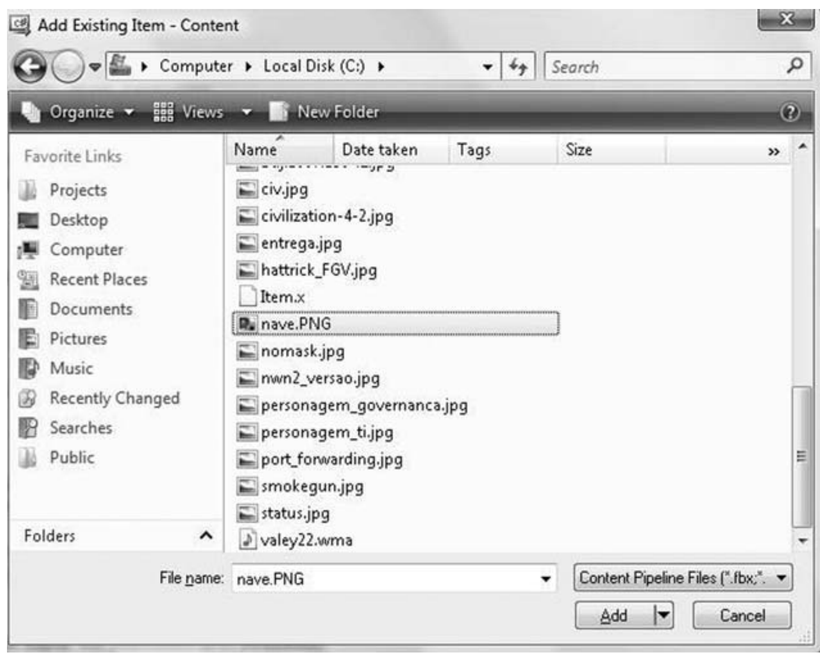


Figura 6.5: Procurando e selecionando o arquivo de imagem da espaçonave.
Fonte: Visual C#.

Atividade prática 2

Procure na internet uma imagem para a espaçonave do jogador e acrescente-a ao seu projeto de jogo, seguindo os três passos descritos anteriormente. O ideal é que essa imagem seja do tipo jpeg (.jpg) ou *portable network graphics* (.png). *Dica*: para encontrar mais facilmente a imagem, utilize o aplicativo Cooliris: http://www.cooliris.com/.

Agora que você já tem a imagem da espaçonave no projeto de jogo, altere o código do projeto para carregar essa imagem e exibir a espaçonave na tela.

Passo 1: Para carregar a imagem da espaçonave no projeto do jogo, chamaremos o método **CarregarTextura** da classe Espaçonave. Isso mesmo, o que acabamos de criar!

O melhor lugar para colocar essa chamada é dentro do método **LoadContent** do projeto de jogo, onde todo o conteúdo do jogo é carregado. Veja como ficou:

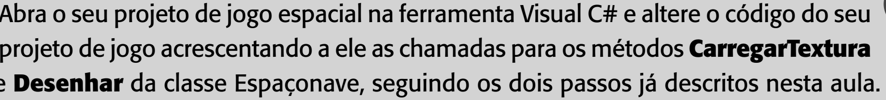
```
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used
    to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game
    content here
    NaveJogador.CarregarTextura(graphics.
    GraphicsDevice,"../../../Content/Imagens/nave.
    PNG");
}
```

Passo 2: O último passo é fazer a espaçonave aparecer na tela, desenhando-a. Para isso, utilizaremos o método **Desenhar** da classe Espaçonave, que também criamos há pouco.

O local ideal para colocar essa chamada é dentro do método **Draw** do projeto de jogo, onde todos os objetos do jogo são desenhados. Veja como ficou:

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

    // TODO: Add your drawing code here
    NaveJogador.Desenhar(spriteBatch);

    base.Draw(gameTime);
}
```

Atividade prática 3



Abra o seu projeto de jogo espacial na ferramenta Visual C# e altere o código do seu projeto de jogo acrescentando a ele as chamadas para os métodos **CarregarTextura** e **Desenhar** da classe Espaçonave, seguindo os dois passos já descritos nesta aula.

Quer ver a espaçonave aparecer na tela do seu jogo? Para executar o código do seu projeto de jogo na ferramenta Visual C#, basta clicar no botão ▶ , localizado na barra superior de ícones, ou então apertar a tecla F5. A tela de jogo que vai aparecer será semelhante à **Figura 6.6**.

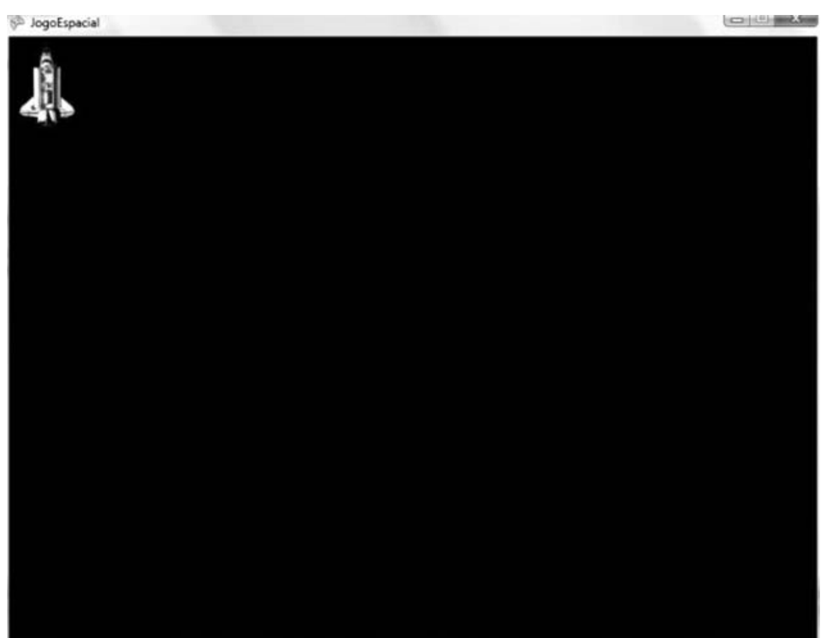


Figura 6.6: Tela do jogo após a execução do código do projeto de jogo.
Fonte: Visual C#.

Atividade prática 4

Abra o seu projeto de jogo espacial na ferramenta Visual C# e execute-o para ver se a espaçonave vai aparecer na tela do jogo. Caso apareça, altere o código do jogo para fazer a espaçonave do jogador aparecer em outra posição da tela que não seja no canto superior esquerdo. Caso a espaçonave não apareça na tela, reveja o seu código do projeto de jogo e procure ajuda no fórum.

Movimentando a espaçonave pela tela do jogo

A espaçonave do jogador já aparece na tela do jogo? Excelente! Mas ela ainda permanece imóvel. Precisamos cuidar agora da movimentação.

Você se lembra do método **Mover**, que já existia na nossa classe Espaçonave?

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

Exatamente, precisamos programá-lo para que a nossa espaçonave saia da inércia e ganhe movimentação pela tela do jogo.

Repare que o método **Mover** recebe um parâmetro, que é a direção para a qual a espaçonave deve se movimentar. Veja na **Figura 6.7** quais são as possíveis direções para onde a espaçonave pode se mover:

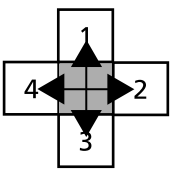
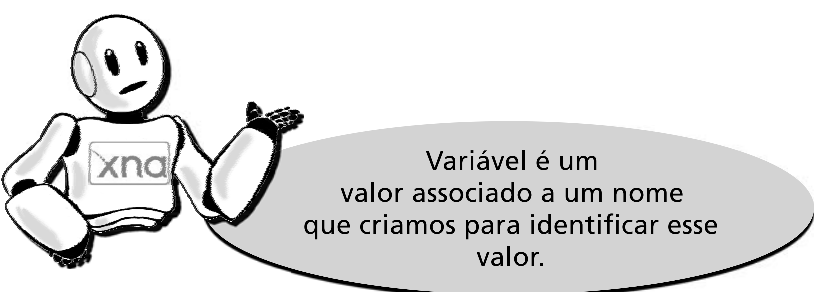


Figura 6.7: Direções de movimentação.
Fonte: Microsoft Word.

A direção 1, por exemplo, significa ir para cima. A direção 2, para a direita; 3 para baixo e 4 para a esquerda.

Passo 1: Para realizar a movimentação da espaçonave, criaremos dentro do método Mover duas variáveis temporárias, que são **deslocamento_x** e **deslocamento_y**.



As variáveis temporárias **deslocamento_x** e **deslocamento_y** armazenarão temporariamente números que significam o quanto a espaçonave irá se deslocar dentro da tela do jogo nos eixos x (horizontal) e y (vertical).

Suponha que desejemos deslocar a espaçonave para baixo, ou seja, na direção 3. Quais serão os valores que as variáveis **deslocamento_x** e **deslocamento_y** armazenarão para executar esse deslocamento?

Analisando o desenho, você pode ver que o deslocamento para baixo (direção 3) é um deslocamento vertical. Nesse caso, a variável **deslocamento_x** irá ficar com o valor 0, pois não haverá deslocamento horizontal. Já a variável **deslocamento_y** irá assumir o valor correspondente à velocidade atual da espaçonave, que é +5. Se fosse um deslocamento para cima, a variável **deslocamento_y** assumiria deslocamento negativo, ou seja, -5. Veja então como ficará o código do método **Mover** da classe Espaçonave:

```
public void Mover(int direcao)
{
    // Variáveis temporárias
    int deslocamento_x, deslocamento_y;
    deslocamento_x = 0;
    deslocamento_y = 0;

    // Deslocamento para cima
    if (direcao == 1)
```

```
{
deslocamento_y = - _velocidade;
}

// Deslocamento para a direita
if (direcao == 2)
{
deslocamento_x = + _velocidade;
}

// Deslocamento para baixo
if (direcao == 3)
{
deslocamento_y = + _velocidade;
}

// Deslocamento para a esquerda
if (direcao == 4)
{
deslocamento_x = - _velocidade;
}

// Executando o movimento
_posicaox = _posicaox + deslocamento_x;
_posicaoy = _posicaoy + deslocamento_y;
}
```

Após os comandos **if**, o movimento é executado, ou seja, os valores das posições x e y da espaçonave são alterados conforme o deslocamento.

Repare que essas movimentações todas não verificam algo bastante importante: se a espaçonave saiu da tela!

Passo 2: Podemos ajustar o método **Mover** para realizar essa verificação também. Veja como fica:

```
 public void Mover(int direcao, int p_posicaox_
 maxima, int p_posicaoy_maxima)
{
   // Variáveis temporárias
   int deslocamento_x, deslocamento_y;
   deslocamento_x = 0;
   deslocamento_y = 0;

   // Deslocamento para cima
   if (direcao == 1)
   {
   deslocamento_y = - _velocidade;
   }

   // Deslocamento para a direita
   if (direcao == 2)
   {
   deslocamento_x = + _velocidade;
   }

   // Deslocamento para baixo
   if (direcao == 3)
   {
   deslocamento_y = + _velocidade;
   }

   // Deslocamento para a esquerda
   if (direcao == 4)
   {
   deslocamento_x = - _velocidade;
   }

   // Validando o movimento da espaçonave
   if (_posicaoy + deslocamento_y >= 0 &&
       _posicaox + deslocamento_x >= 0 &&
       _posicaox + deslocamento_x <=
       p_posicaox_maxima &&
```

```
 _posicaoy + deslocamento_y <= p_posicaoy_
 maxima)
    {
// Executando o movimento
  _posicaox = _posicaox + deslocamento_x;
  _posicaoy = _posicaoy + deslocamento_y;
    }
}
```

Para realizar essa verificação, foram acrescentados mais dois parâmetros de entrada ao método: a posição x e a posição y máximas da tela. Dessa forma, antes de executar o movimento, o código testa se o deslocamento irá ultrapassar os valores mínimos e máximos de x e y da tela e só realiza a movimentação da espaçonave se o movimento estiver correto, impedindo que a nave saia da tela do jogo.

Atividade prática 5

Seguindo os dois passos descritos, abra seu projeto de jogo espacial na ferramenta Visual C# e altere o código da classe Espaçonave dentro do seu projeto de jogo para acrescentar a ele o código do método **Mover**.

A movimentação já está programada dentro da classe Espaçonave. Precisamos agora programar o jogo para que a movimentação seja realizada quando apertarmos as setas do teclado.

Passo 1: Acrescente aos atributos do projeto de jogo um leitor para as teclas do teclado, o KeyboardState.

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

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

// Definindo um leitor de teclas do teclado
KeyboardState teclado;
```

Passo 2: Altere também o método de atualização do jogo (**Update**). Dentro dele colocaremos toda a lógica de movimentação da espaçonave, conforme as setas do teclado forem sendo pressionadas. Vamos utilizar o comando **Keyboard.GetState** para verificar o estado do teclado, ou seja, recuperar quais teclas foram pressionadas pelo jogador. Veja como ficou:

```
protected override void Update(GameTime gameTime)
{
       // Allows the default game to exit on Xbox 360 and
       Windows
       if (GamePad.GetState(PlayerIndex.One).Buttons.Back
       == ButtonState.Pressed)
       this.Exit();

        // TODO: Add your update logic here
        teclado = Keyboard.GetState();
        if (teclado.IsKeyDown(Keys.Up))
        {
           NaveJogador.Mover(1, this.Window.ClientBounds.
           Width, this.Window.ClientBounds.Height);
        }
        if (teclado.IsKeyDown(Keys.Right))
        {
            NaveJogador.Mover(2, this.Window.ClientBounds.
            Width, this.Window.ClientBounds.Height);
        }
        if (teclado.IsKeyDown(Keys.Down))
        {
          NaveJogador.Mover(3, this.Window.ClientBounds.
          Width, this.Window.ClientBounds.Height);
```

```
            }
            if (teclado.IsKeyDown(Keys.Left))
            {
               NaveJogador.Mover(4,  this.Window.ClientBounds
               .Width, this.Window.ClientBounds.Height);
            }

            base.Update(gameTime);
        }
```

Repare também que são realizados diversos testes (**if**) para verificar se cada uma das setas do teclado foi pressionada. Para testar se o jogador apertou a seta para cima, por exemplo, usamos o comando **IsKeyDown(Keys.Up)**. Caso alguma das setas do teclado tenha sido apertada, veja que o método **Mover** da espaçonave do jogador é acionado e a direção do movimento é enviada para ele junto com os valores máximos das posições x e y da tela. Esses valores máximos são obtidos com os comandos **Window.ClientBounds.Width** (largura da tela) e **Window.ClientBounds.Height** (altura da tela).

Atividade prática 6

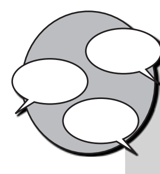
Abra seu projeto de jogo espacial na ferramenta Visual C# e altere o código do seu projeto de jogo acrescentando a ele o atributo leitor das teclas do teclado (**KeyboardState**) e as chamadas para o método **Mover** da classe Espaçonave, seguindo os dois passos descritos. Depois, execute seu projeto de jogo e pressione as setas para verificar se a espaçonave está se movimentando corretamente.

Atividade prática 7

Com base no conhecimento adquirido para criar e desenhar espaçonaves, abra seu projeto de jogo espacial na ferramenta Visual C# e crie dentro do seu jogo uma espaçonave inimiga chamada *Soldado do Espaço* e programe sua movimentação sempre na direção 3 (andar para baixo).

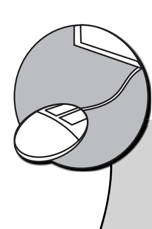
Atividade prática 8

Existe uma pequena falha no código de movimentação da classe Espaçonave. Quando a espaçonave se movimenta para o canto direito ou inferior da tela, ela ainda some. Tente melhorar o método **Mover** da classe Espaçonave para que isso não aconteça. **Dica**: Utilize a altura e a largura da textura da espaçonave (**_textura.Width** e **_textura.Height**).



INFORMAÇÕES SOBRE FÓRUM

Você teve alguma dificuldade para exibir ou movimentar a sua espaçonave no jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.



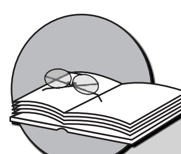
Atividade *online*

Agora que você já está com o seu código de projeto do jogo ajustado para exibir e movimentar a espaçonave, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- Para exibir as espaçonaves dentro do jogo espacial, utilizamos o conceito de texturas. Textura é uma imagem utilizada para revestir os objetos criados em duas ou três dimensões dentro de um projeto de jogo. Neste caso, estaremos revestindo um retângulo com a imagem de uma espaçonave no formato jpeg ou PNG.

- A imagem da espaçonave pode ser exibida na tela do jogo. Para tal, é necessário carregá-la dentro de um **SpriteBatch**, ou seja, um conjunto de *sprites*, que são áreas de tela específicas para o carregamento de texturas.
- Em seguida, será necessário criar os métodos **CarregarTextura** e **Desenhar** na classe **Espaçonave** para que possamos carregar a imagem da espaçonave e exibi-la na tela do jogo posteriormente. Também será necessário alterar o método **Draw** do projeto de jogo para chamar o método Desenhar da classe Espaçonave durante a execução do jogo.
- Podemos programar a movimentação da espaçonave pela tela. Para isso, precisaremos alterar o método **Mover** da classe Espaçonave, de forma que ele modifique as posições x e y da espaçonave de acordo com a direção do movimento e a velocidade atual da espaçonave.
- Também será necessário acrescentar um leitor de teclas do teclado, o **KeyboardState**, para verificar quais teclas foram pressionadas durante o jogo. Além disso, o método **Update** do jogo precisa ser modificado para levar em conta quais setas do teclado foram pressionadas e chamar o método **Mover** da classe Espaçonave para executar a movimentação.
- Você pode executar um projeto de jogo utilizando o ícone , localizado na barra superior de ícones, ou apertando a tecla F5.



Informação sobre a próxima aula

Na próxima aula, você verá como acrescentar tiros e um fundo de tela ao jogo.

AULA 7

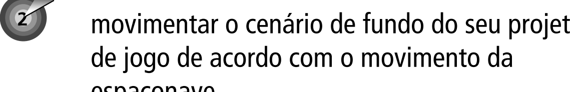
Acrescentando tiros e um cenário espacial ao seu jogo

Metas da aula

Atirar com a espaçonave e movimentar o cenário de fundo do jogo.

objetivos

Ao final desta aula, você deverá ser capaz de:



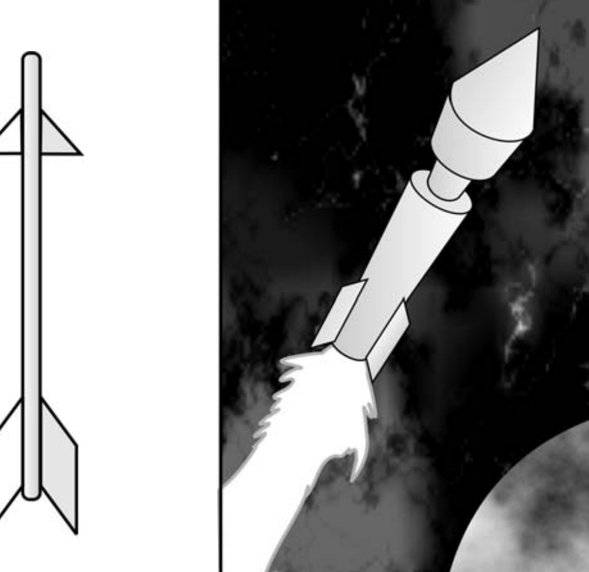
1. atirar com a espaçonave do seu projeto de jogo;
2. movimentar o cenário de fundo do seu projeto de jogo de acordo com o movimento da espaçonave.

Pré-requisitos

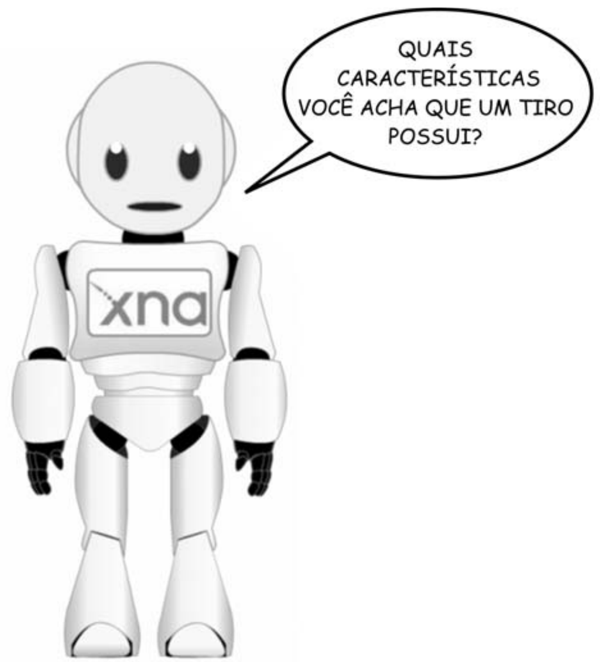
Estar familiarizado com a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na Aula 3; reconhecer os conceitos de classe e objeto, abordados na Aula 4; ter o seu projeto de jogo com a ferramenta Visual C# atualizado para exibir e movimentar a espaçonave do jogador, conforme visto na Aula 6.

INTRODUÇÃO

Na aula anterior, aprendemos como exibir e movimentar as espaçonaves dentro do jogo. Agora podemos acrescentar alguns detalhes para torná-lo mais interessante. Que tal fazermos a espaçonave do jogador atirar? Assim, ela poderá enfrentar seus inimigos, os soldados do espaço.



Figuras 7.1, 7.2 e 7.3: Exemplos de tiros.



Veja três características do tiro:

- potência;
- direção;
- velocidade.

Além disso, precisamos armazenar informações sobre a localização e o visual do tiro, para fazê-lo aparecer dentro da tela do nosso jogo.

CRIAÇÃO DA CLASSE TIRO

Lembra da classe Espaçonave, que criamos na Aula 4? Vamos agora criar uma classe para os tiros disparados no jogo. Basta seguir estes passos:

Passo 1: Criar os atributos e o método de criação da classe **Tiro**.

A classe **Tiro** possuirá os seguintes **atributos: potência, direção, velocidade, posição** e **imagem.**

Veja como ficou a nossa classe Tiro:

```
public class Tiro
{
        private int _potencia;
        private int _direcao;
        private int _velocidade;
        private int _posicaox;
        private int _posicaoy;
        private Texture2D _textura;

          public Tiro(int p_potencia, int p_direcao, int
p_velocidade, int p_posicaox, int p_posicaoy)
        {
            _potencia = p_potencia;
            _direcao = p_direcao;
            _velocidade = p_velocidade;
            _posicaox = p_posicaox;
            _posicaoy = p_posicaoy;
        }
}
```

Repare no método **Tiro**, que é o método de criação da classe **Tiro**. Observe que, ao contrário da espaçonave, que não precisava receber nenhuma informação especial durante sua criação, cada tiro requer as seguintes informações no momento da sua criação: potência do tiro (**p_potencia**), direção do tiro (**p_direcao**), velocidade do tiro (**p_velocidade**) e posição inicial do tiro (**p_posicaox** e **p_posicaoy**).

Passo 2: Criar o método **Mover** para a classe **Tiro.**

Figura 7.4: O tiro em movimento.

Assim como a espaçonave, o tiro também não fica parado. Tente identificar quais ações um tiro vai desempenhar no jogo:

- mover;
- explodir.

Repare que nem todos os tiros irão explodir. Dessa forma, vamos nos concentrar primeiro em criar um método para movimentar os tiros: o método **Mover** da classe **Tiro**. Ele é uma versão simplificada do método **Mover** da classe **Espaçonave**.

Observe que o tiro possui sua própria direção. Assim, o método Mover da classe Tiro não receberá a direção como parâmetro. Também não precisaremos verificar se o tiro "saiu da tela", como fazíamos no método **Mover** da classe **Espaçonave**. Veja como ficou:

```
public void Mover()
{
    // Variáveis temporárias
    int deslocamento_x, deslocamento_y;
    deslocamento_x = 0;
    deslocamento_y = 0;

    // Deslocamento para cima
    if (_direcao == 1)
    {
        deslocamento_y = - _velocidade;
    }

    // Deslocamento para a direita
    if (_direcao == 2)
    {
        deslocamento_x = + _velocidade;
    }

    // Deslocamento para baixo
    if (_direcao == 3)
    {
        deslocamento_y = + _velocidade;
    }
```

```
        // Deslocamento para a esquerda
        if (_direcao == 4)
        {
            deslocamento_x = - _velocidade;
        }

        // Executando o movimento
        _posicaox = _posicaox + deslocamento_x;
        _posicaoy = _posicaoy + deslocamento_y;
    }
```

Passo 3: Criar os demais métodos da classe Tiro.

Perceba que o tiro, além de se mover, também precisa ser exibido em uma determinada posição da tela do jogo. Para que isso aconteça, vamos construir os seguintes métodos:

- **PosicaoX** e **PosicaoY**: apenas para retornar à posição atual (coordenadas x e y) do tiro.
- **CarregarTextura**: assim como na classe Espaçonave, este método serve para atribuir uma imagem ao tiro.
- **Textura**: apenas para retornar à imagem atual que foi associada ao tiro.
- **Desenhar**: para fazer o tiro aparecer na tela do jogo.

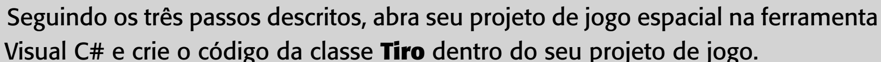
Veja a seguir como foram construídos esses métodos:

```
public int PosicaoX()
    {
        return _posicaox;
    }

    public int PosicaoY()
    {
        return _posicaoy;
    }

    public void CarregarTextura(GraphicsDevice p_
    dispositivo, string p_local)
```

```
 {
    _textura = Texture2D.FromFile(p_dispositivo,
    p_local);
}

public Texture2D Textura()
{
    return _textura;
}

public void Desenhar(SpriteBatch p_sprite)
{

    p_sprite.Begin();
    p_sprite.Draw(_textura, new Rectangle(
    _posicaox, _posicaoy, _textura.Width,
    _textura.
    Height), Color.White);
    p_sprite.End();
}
```

Pronto! Nossa classe **Tiro** já está completa. Observe que os métodos **CarregarTextura** e **Desenhar** são idênticos aos da classe **Espaçonave.**

Atividade prática 1



Seguindo os três passos descritos, abra seu projeto de jogo espacial na ferramenta Visual C# e crie o código da classe **Tiro** dentro do seu projeto de jogo.

Acrescentando os tiros ao jogo

Agora que a classe **Tiro** já foi construída, vamos ver como incorporar os tiros ao nosso projeto de jogo.

Pense sobre esse assunto e analise as seguintes questões:

- Quem vai controlar os tiros?
- Quem irá disparar os tiros?
- Os tiros existirão para sempre?

Encontrou as respostas? Vamos a elas, então:

Questão 1: Quem vai controlar os tiros?

Resposta: O jogo. Vamos criar dentro do jogo uma lista para guardar todos os tiros que já foram disparados. Assim poderemos controlar a movimentação dos tiros no jogo.

```
namespace JogoEspacial
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1: Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

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

    // Definindo um leitor de teclas do teclado
    KeyboardState teclado;

   // Criando uma lista de tiros disparados no jogo
      List<Tiro> TirosDisparados;
```

Para criar uma lista, usamos o comando **List<tipo>**, em que **<tipo>** define quais são os elementos que farão parte dessa lista. No nosso caso, criamos uma lista de elementos do tipo **Tiro** e chamamos a lista de **TirosDisparados**.

Precisamos também inicializar a lista de tiros disparados no método de inicialização (**Initialize**) do jogo:

```
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");

// Inicializando a lista de tiros do jogo
TirosDisparados = new List<Tiro>();

     base.Initialize();
}
```

Repare que ainda falta controlarmos a movimentação de todos os tiros que existem no jogo. Podemos fazer isso no método de atualização do jogo (**Update**). Para isso, precisaremos movimentar cada um dos tiros do jogo percorrendo a lista de tiros disparados. Veja como:

```
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
          teclado = Keyboard.GetState();
          if (teclado.IsKeyDown(Keys.Up))
          {
     NaveJogador.Mover(1, this.Window.ClientBounds.Width,
     this.Window.ClientBounds.Height);
          }
           if (teclado.IsKeyDown(Keys.Right))
          {
```

```
    NaveJogador.Mover(2, this.Window.
    ClientBounds.Width, this.Window.ClientBounds.
    Height);
  }
  if (teclado.IsKeyDown(Keys.Down))
  {
    NaveJogador.Mover(3, this.Window.
    ClientBounds.Width, this.Window.ClientBounds.
    Height);
  }
  if (teclado.IsKeyDown(Keys.Left))
  {
    NaveJogador.Mover(4, this.Window.
    ClientBounds.Width, this.Window.ClientBounds.
    Height);
  }

    // Movendo os tiros do jogo
    foreach (Tiro oTiro in TirosDisparados.
    GetRange(0,TirosDisparados.Count))
{
  oTiro.Mover();
}

  base.Update(gameTime);
}
```

Perceba que utilizamos um novo comando para percorrer a lista de tiros, o comando **foreach**. Para cada tiro na lista de tiros disparados, chamamos o método Mover desse tiro (**oTiro.Mover**) para que ele atualize sua posição e se movimente pela tela do jogo.

O método de exibição do jogo (**Draw**) também precisa ser atualizado. Acrescente uma instrução para desenhar cada um dos tiros do jogo, de forma semelhante ao que fizemos anteriormente para movimentar os tiros:

```
protected override void Draw(GameTime
gameTime)
```

```
{
    GraphicsDevice.Clear(Color.Black);
      // TODO: Add your drawing code here
    NaveJogador.Desenhar(spriteBatch);

    // Desenhando os tiros do jogo
      foreach (Tiro oTiro in TirosDisparados)
  {
    oTiro.Desenhar(spriteBatch);
  }

    base.Draw(gameTime);
}
```

Questão 2: Quem irá disparar o tiro?

Resposta: A espaçonave, quando o jogador apertar algum botão.

Certo! Vamos então programar o método **Atirar** da classe **Espaçonave** para que ela produza os tiros.

```
public void Atirar(GraphicsDevice p_
dispositivo_grafico, string p_local_
textura_tiro, List<Tiro> p_lista_
tiros)
{
   // Cria um tiro
   Tiro oTiro;
    oTiro = new Tiro(_potencia_tiro,
    1, _velocidade_tiro, _posicaox,
    _posicaoy);
```

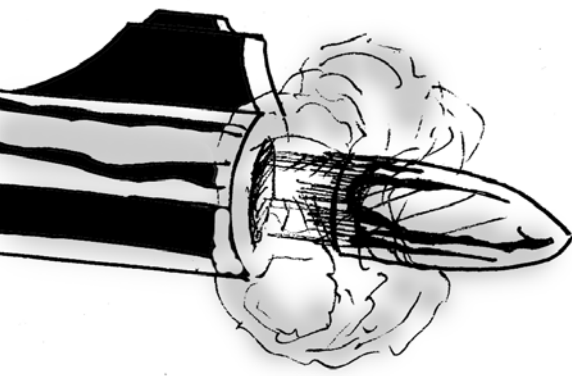


Figura 7.5: O ato de atirar.

```
        // Carrega a textura do tiro
          oTiro.CarregarTextura(p_dispositivo_grafico,
          p_local_textura_tiro);

      // Acrescenta o tiro na lista de tiros do jogo
          p_lista_tiros.Add(oTiro);
    }
```

Observe que o método **Atirar** recebe alguns parâmetros importantes:

- **p_dispositivo_grafico**: o dispositivo gráfico do jogo, no qual será exibido o tiro;
- **p_local_textura_tiro**: o local onde se encontra a imagem que será usada para o tiro;
- **p_lista_tiros**: a lista de tiros disparados no jogo.

Em seguida, ele cria um tiro chamado **oTiro**, e na sua criação define a potência (**_potencia_tiro**), a direção (1), a velocidade (**_velocidade_tiro**) e a posição inicial do tiro (**_posicaox e _posicaoy**). Repare que a posição inicial do tiro é a mesma da espaçonave que atirou.

Por fim, ele carrega a imagem do tiro como textura utilizando o método **oTiro.CarregarTextura** e adiciona o tiro criado à lista de tiros do jogo usando o comando **Add**.

Adicione também à sua classe Espaçonave um valor inicial para a potência do tiro. Isso pode ser feito ajustando o método de criação da classe Espaçonave:

```
public Espaconave()
{
    _energia = 50;
    _velocidade = 5;
    _velocidade_tiro = 5;
    _posicaox = 10;
    _posicaoy = 10;
 _potencia_tiro = 10;
}
```

Ainda falta um detalhe: o tiro da espaçonave do jogador só é disparado quando o jogador aperta algum botão.

Figura 7.6: Pressionando a barra de espaços.

Vamos configurar a barra de espaços para ser nosso botão de disparar os tiros. Isso pode ser feito dentro do método de atualização do jogo (**Update**). Observe:

```
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
            teclado = Keyboard.GetState();
            if (teclado.IsKeyDown(Keys.Up))
            {
             NaveJogador.Mover(1, this.Window.
             ClientBounds.Width, this.Window.ClientBounds.
             Height);
            }
            if (teclado.IsKeyDown(Keys.Right))
```

```
{
        NaveJogador.Mover(2, this.Window.
        ClientBounds.Width, this.Window.
        ClientBounds.Height);
}
if (teclado.IsKeyDown(Keys.Down))
{
        NaveJogador.Mover(3, this.Window.
        ClientBounds.Width, this.Window.
        ClientBounds.Height);
}
if (teclado.IsKeyDown(Keys.Left))
{
        NaveJogador.Mover(4, this.Window.
        ClientBounds.Width, this.Window.
        ClientBounds.Height);
}
//Apertou o tiro
if (teclado.IsKeyDown(Keys.Space))
{
NaveJogador.Atirar(graphics.GraphicsDevice,
"../../../Content/Imagens/tiro.png",
TirosDisparados);
}

// Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados)
{
    oTiro.Mover();
}

base.Update(gameTime);
}
```

Utilizamos o comando **IsKeyDown**(**Keys.Space**) para verificar se a barra de espaços foi pressionada. Caso tenha sido, chamamos o método **Atirar** da espaçonave do jogador, passando para ele o dispositivo gráfico do jogo, o local da imagem do tiro e a lista de tiros do jogo.

Repare também que utilizamos a imagem **tiro.png** para os tiros da espaçonave do jogador. Esse arquivo de imagem precisa ser acrescentado ao projeto de jogo, da mesma forma como fizemos na Aula 6 para a imagem da espaçonave do jogador.

Questão 3: Os tiros existirão para sempre?

Resposta: Não. É importante retirar da lista todos os tiros que já "saíram da tela do jogo".

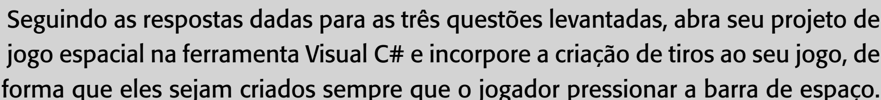
Para limpar a lista de tiros, mantendo nela apenas aqueles que ainda estão dentro da tela do jogo, podemos acrescentar um teste ao método de atualização do jogo (**Update**), para verificar se cada um dos tiros já ultrapassou os limites da tela e removê-lo da lista de tiros do jogo. Veja como ficou:

```
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
            teclado = Keyboard.GetState();
            if (teclado.IsKeyDown(Keys.Up))
            {
             NaveJogador.Mover(1, this.Window.
             ClientBounds.Width, this.Window.ClientBounds.
             Height);
            }
            if (teclado.IsKeyDown(Keys.Right))
            {
             NaveJogador.Mover(2, this.Window.
             ClientBounds.Width, this.Window.ClientBounds
             Height);
            }
```

```
 if (teclado.IsKeyDown(Keys.Down))
{
        NaveJogador.Mover(3, this.Window.
        ClientBounds.Width, this.Window.
        ClientBounds.Height);
}
if (teclado.IsKeyDown(Keys.Left))
{
        NaveJogador.Mover(4, this.Window.
        ClientBounds.Width, this.Window.
        ClientBounds.Height);
}
if (teclado.IsKeyDown(Keys.Space))
 //Apertou o tiro
{
 NaveJogador.Atirar(graphics.
 GraphicsDevice, "../../../Content/Imagens/
 tiro.png", TirosDisparados);
}

// Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados
 GetRange(0,TirosDisparados.Count))
{
    oTiro.Mover();

// Remove os tiros que saíram da tela do jogo
if (oTiro.PosicaoX() < 1 ||
    oTiro.PosicaoY() < 1 ||
    oTiro.PosicaoX() >
this.Window.ClientBounds.Width - 1 ||
    oTiro.PosicaoY() >
this.Window.ClientBounds.Height - 1)
   {
        TirosDisparados.Remove(oTiro);
    }
}
}
```

Repare que colocamos a verificação logo após a movimentação do tiro (**oTiro. Mover**). Testamos se a posição do tiro (coordenadas x e y) é menor que 1 ou maior que a largura (**this.Window.ClientBounds. Width**) ou a altura (**this.Window. ClientBounds.Height**) da tela. Nesse caso, retiramos o tiro da lista de tiros do jogo por meio do comando **Remove(oTiro)**.

Atividade prática 2



Seguindo as respostas dadas para as três questões levantadas, abra seu projeto de jogo espacial na ferramenta Visual C# e incorpore a criação de tiros ao seu jogo, de forma que eles sejam criados sempre que o jogador pressionar a barra de espaço.

Atividade prática 3

Existe uma pequena falha no jogo que faz com que vários tiros sejam criados de uma só vez quando o jogador aperta apenas uma vez a barra de espaço. Ajuste o código do jogo para que, a cada vez que o jogador aperte a barra de espaço, apenas um único tiro seja gerado.

Dicas:

- Utilize o comando **IsKeyUp**(**Keys.Space**) para verificar quando a barra de espaço deixou de ser apertada.
- Crie no jogo um atributo do tipo verdadeiro/falso (**boolean**) para controlar exatamente quando a barra de espaço já foi pressionada ou solta.

Atividade prática 4

Existe outro pequeno problema no jogo. Quando o tiro é criado, ele sai do canto esquerdo da espaçonave. Modifique o código do jogo para fazer com que o tiro saia do meio da espaçonave.

Dicas:

- Crie na classe **Tiro** os métodos para atualizar a posição do tiro (coordenadas x e y).
- Utilize o comando **_textura.Width** para obter a largura da textura da espaçonave.

Acrescentando um cenário de fundo ao jogo

Veja agora como acrescentar um cenário de fundo móvel ao jogo, de forma a dar a sensação de movimento constante da nave. Para tal, siga os passos descritos a seguir.

Passo 1: Acrescentar dois novos atributos ao nosso jogo:

- uma textura para usar como cenário de fundo, que chamaremos de **cenário_fundo**;
- a posição vertical atual deste cenário de fundo do jogo, que vamos chamar de **posicaoy_cenario_fundo**;

Observe agora os atributos do jogo:

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

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

    // Definindo um leitor de teclas do teclado
    KeyboardState teclado;

    // Criando uma lista de tiros disparados no jogo
    List<Tiro> TirosDisparados;

    // Atributos do cenário de fundo do jogo
    Texture2D cenario_fundo;
    int posicaoy_cenario_fundo;
```

Passo 2: Alterar o método de inicialização do jogo (**Initialize**) para configurar uma posição vertical inicial para o cenário de fundo do jogo. Inicialmente, colocaremos o valor zero, para que o fundo seja projetado inicialmente cobrindo toda a tela, do início ao fim.

```
    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");

// Inicializando a lista de tiros do jogo
TirosDisparados = new List<Tiro>();

// Definindo a posição inicial do cenário de fundo
posicaoy_cenario_fundo = 0;

apertou_tiro = false;

base.Initialize();
}
```

Passo 3: Ajustar no método de carregamento do conteúdo (**LoadContent**) para carregar a textura do cenário de fundo do jogo:

```
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
NaveJogador.CarregarTextura(graphics.GraphicsDevice,
"../../../Content/Imagens/nave.PNG");

// Carregando o cenário de fundo do jogo
cenario_fundo = Texture2D.FromFile(
graphics.GraphicsDevice,
"../../../Content/Imagens/espaco.jpg");
}
```

Repare também que utilizamos a imagem **espaco.jpg** como o cenário de fundo do jogo. Esse arquivo de imagem precisa ser acrescentado ao projeto de jogo da mesma forma como fizemos na Aula 6, para a imagem da espaçonave do jogador.

Passo 4: Mover o cenário de fundo do jogo conforme a movimentação da espaçonave. Para tal, vamos atualizar constantemente a posição y do cenário de fundo no método de atualização do jogo (**Update**). Dessa forma, a cada vez o cenário de fundo será desenhado numa posição vertical diferente da anterior, trazendo a sensação de movimento. Veja:

```
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
      teclado = Keyboard.GetState();
      if (teclado.IsKeyDown(Keys.Up))
      {
       NaveJogador.Mover(1, this.Window.
       ClientBounds.Width, this.Window.ClientBounds.
       Height);
      }
      if (teclado.IsKeyDown(Keys.Right))
      {
       NaveJogador.Mover(2, this.Window.
       ClientBounds.Width, this.Window.ClientBounds.
       Height);
      }
      if (teclado.IsKeyDown(Keys.Down))
      {
       NaveJogador.Mover(3, this.Window.
       ClientBounds.Width, this.Window.ClientBounds.
       Height);
       }
       if (teclado.IsKeyDown(Keys.Left))
       {
```

```
NaveJogador.Mover(4, this.Window.
ClientBounds.Width, this.Window.
ClientBounds.
Height);
}
if (teclado.IsKeyDown(Keys.Space) &&
!apertou_tiro) //Apertou o tiro
{
NaveJogador.Atirar(graphics.
GraphicsDevice, "../../../Content/Imagens/
tiro.png", TirosDisparados);
apertou_tiro = true;
}
if (teclado.IsKeyUp(Keys.Space) &&
apertou_tiro)
{
apertou_tiro = false;
}

// Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados
GetRange(0,TirosDisparados.Count))
{
oTiro.Mover();

// Remove os tiros que saíram da tela do jogo
    if (oTiro.PosicaoX() < 1 ||
        oTiro.PosicaoY() < 1 ||
        oTiro.PosicaoX() > this.Window.
        ClientBounds.Width - 1 ||
        oTiro.PosicaoY() > this.Window.
        ClientBounds.Height - 1)
    {
        TirosDisparados.Remove(oTiro);
    }
}
```

```
        // Movimentando o cenário de fundo do jogo
        posicaoy_cenario_fundo += NaveJogador.Velocidade()
      / 2;
        if (posicaoy_cenario_fundo >= this.Window.
        ClientBounds.Height)
            posicaoy_cenario_fundo = 0;
}
```

Repare que existe um pequeno teste que faz a posição y do cenário de fundo retornar a zero quando o valor desta excede a altura da tela de jogo.

Passo 5: Ajustar o método de desenhar do jogo (**Draw**) para desenhar a textura do cenário de fundo do jogo antes de todos os outros objetos, de forma que esta fique por baixo das outras texturas. Utilizaremos a posição y definida para a textura de cenário de fundo como posição inicial. Veja como foi feito:

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

  // Desenhando o cenário de fundo do jogo
  spriteBatch.Begin();
  spriteBatch.Draw(cenario_fundo, new Rectangle(0,
  posicaoy_cenario_fundo, this.Window.
  ClientBounds.Width, this.Window.ClientBounds.Height),
  Color.White);
  spriteBatch.Draw(cenario_fundo, new Rectangle(0,
  posicaoy_cenario_fundo - this.Window.
  ClientBounds.Height, this.Window.ClientBounds.Width,
  this.Window.ClientBounds.Height), Color.White);
  spriteBatch.End();

    // TODO: Add your drawing code here
    NaveJogador.Desenhar(spriteBatch);

    // Desenhando os tiros do jogo
    foreach (Tiro oTiro in TirosDisparados)
```

```
            {
                oTiro.Desenhar(spriteBatch);
            }

            base.Draw(gameTime);
        }
```

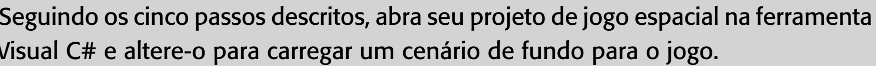
Observe que o cenário de fundo é desenhado duas vezes. O segredo consiste em desenhar uma segunda vez o mesmo cenário de fundo do jogo, posicionando-o de forma a completar o buraco deixado pelo primeiro desenho, devido ao aumento da posição y.

A primeira vez que o cenário de fundo é desenhado, é utilizada a posição y para definir o ponto inicial do desenho. Na segunda vez, utilizamos como ponto inicial a posição y menos a altura da tela (**this.Window.ClientBounds.Height**). Isso fará com que o mesmo cenário de fundo seja desenhado duas vezes, preenchendo todo o espaço da tela e dando a sensação de movimento a cada momento do jogo, já que os cenários de fundo estarão sempre sendo desenhados com base numa posição y diferente.

Figura 7.7: Tela do jogo com tiros e cenário de fundo.
Fonte: Visual C#.

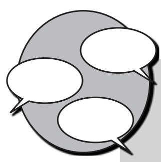
Atividade prática 5

Seguindo os cinco passos descritos, abra seu projeto de jogo espacial na ferramenta Visual C# e altere-o para carregar um cenário de fundo para o jogo.



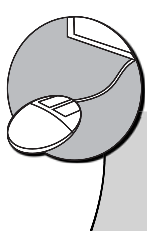
Atividade prática 6

Agora faça com que o cenário de fundo do jogo se mova na posição horizontal, ao invés da vertical. **Dica**: utilize a instrução **this.Window.ClientBounds.Width** para obter a altura da tela do jogo.



INFORMAÇÃO SOBRE FÓRUM

Você teve alguma dificuldade para adicionar tiros ou o cenário de fundo ao jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.



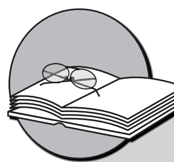
Atividade *online*

Agora que você já está com seu código de projeto do jogo ajustado para adicionar tiros e o cenário de fundo ao jogo, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- Da mesma forma que você criou uma classe para as suas espaçonaves, é preciso criar uma classe para todos os tiros disparados no jogo, a classe **Tiro**. Ela será composta pelos atributos potência, direção, velocidade, posição e imagem e os métodos **Mover**, **CarregarTextura**, **Desenhar**, **Textura**, **PosicaoX** e **PosicaoY**.

- Observe que o tiro possui sua direção própria. Dessa forma, o método **Mover** da classe **Tiro** não receberá a direção como parâmetro. Também não precisaremos verificar se o tiro "saiu da tela", como fazíamos no método **Mover** da classe **Espaçonave**.
- Precisaremos criar uma lista para controlar os tiros que foram disparados dentro do jogo, para que possamos exibi-los e movimentá-los corretamente com o decorrer do tempo de jogo.
- Será necessário programar o método **Atirar** da classe **Espaçonave** para que ela produza os tiros e os acrescente na lista de tiros do jogo.
- É importante acrescentar também uma rotina para retirar da lista de tiros do jogo aqueles que já "saíram da tela do jogo". Isso pode ser feito verificando se cada tiro já ultrapassou os limites da tela do jogo.
- Para movimentar o cenário de fundo do jogo, precisaremos acrescentar dois atributos ao projeto de jogo: a textura **cenario_fundo** e a posição y dessa textura.
- Para dar a sensação de movimento ao cenário de fundo do jogo, basta exibir esse cenário de fundo utilizando uma posição y inicial diferente a cada exibição. É necessário também fazer uma segunda exibição do cenário de fundo, de forma a complementar o espaço vazio que foi deixado pela primeira exibição.



Informação sobre a próxima aula

Na próxima aula, você verá como acrescentar sons ao projeto de jogo. Até lá!

Acrescentando sons ao seu jogo

AULA 8

Meta da aula

Incorporar trilha sonora e som de tiros ao jogo espacial.

objetivos

Ao final desta aula, você deverá ser capaz de:

1. criar um projeto de sons com a ferramenta XACT;
2. acrescentar diversos sons ao seu jogo.

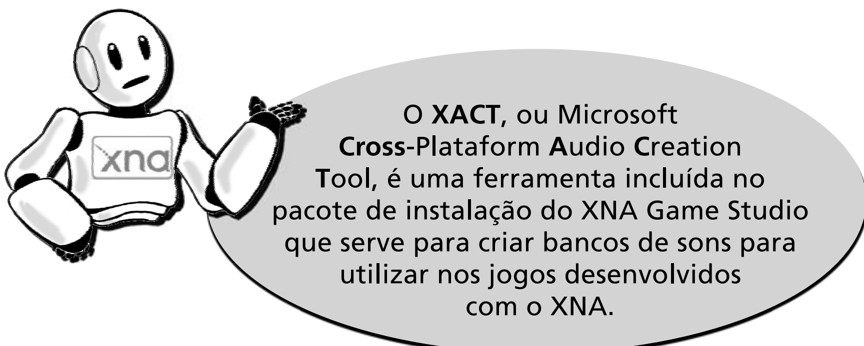
Pré-requisitos

Conhecer a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na Aula 3; ter seu projeto de jogo atualizado conforme o conteúdo da Aula 7, que inclui a produção de tiros pela espaçonave.

INTRODUÇÃO

Na aula anterior, atualizamos nosso projeto de jogo espacial dentro da ferramenta Visual C# acrescentando a produção de tiros e um cenário de fundo para o jogo.

Agora você vai aprender como acrescentar sons aos jogos. Para isso, precisaremos manusear a ferramenta **XACT**. O que significa **XACT**?



Carregue a ferramenta **XACT**. Clique no botão **Iniciar** do Windows e selecione no menu a opção **Programas\Xna Game Studio\Tools\Microsoft Cross-Plataform Audio Creation Tool** (**XACT**). Observe o procedimento na **Figura 8.1**:

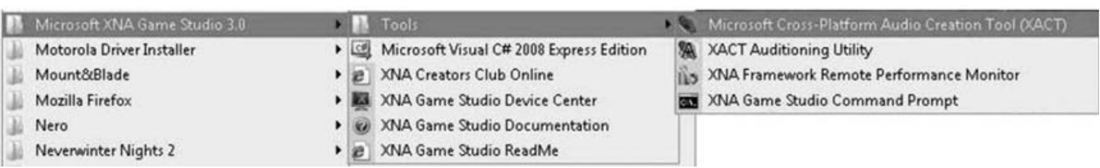


Figura 8.1: Acessando a ferramenta XACT.
Fonte: Windows.

CRIANDO O SEU PROJETO DE SONS NA FERRAMENTA XACT

Veja a tela inicial da ferramenta **XACT** na **Figura 8.2:**

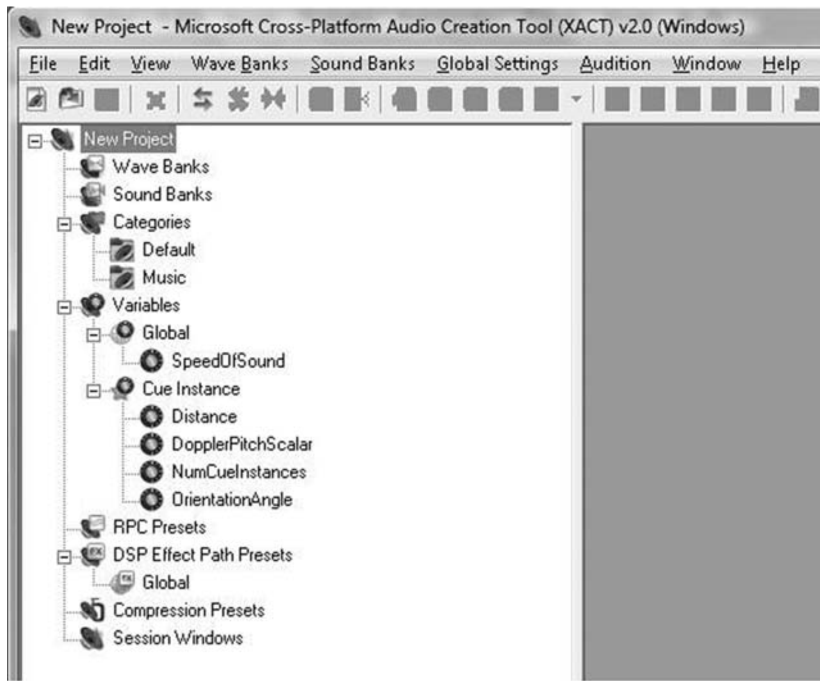


Figura 8.2: Tela inicial da ferramenta XACT.
Fonte: XACT.

Vamos agora iniciar um novo projeto de sons dentro da ferramenta **XACT**. Selecione no menu superior da ferramenta **XACT** a opção **File\New Project**. Será exibida uma janela de busca de arquivos, perguntando o local e o nome do projeto, semelhante à janela exibida na **Figura 8.3:**

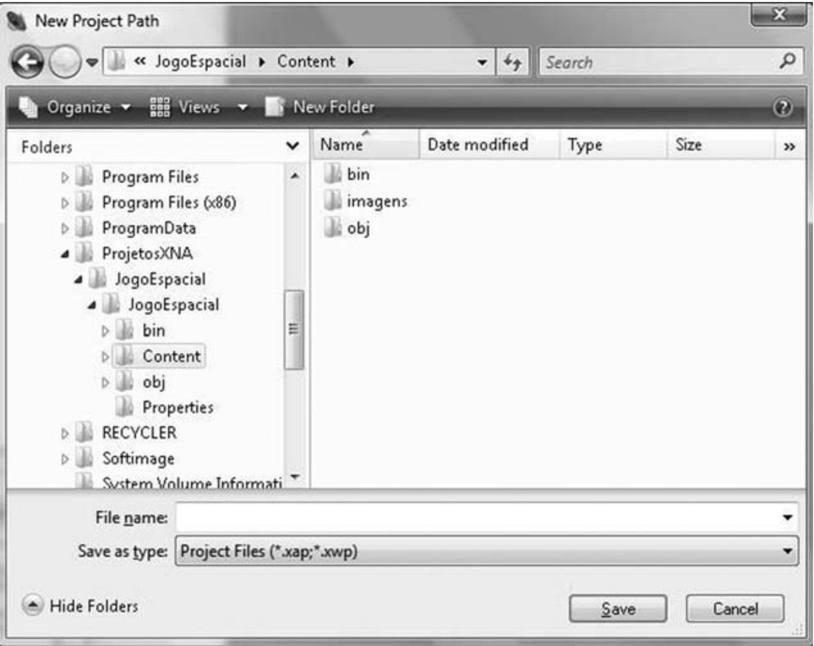


Figura 8.3: Iniciando um novo projeto na ferramenta XACT.
Fonte: XACT.

Procure a pasta onde está salvo o seu jogo espacial (**C:\ProjetosX-NA\ JogoEspacial**). Depois, entre na pasta **JogoEspacial\Content**.

Crie agora, dentro da pasta **Content**, uma nova pasta chamada **audio**. Dentro dela guardaremos todos os arquivos de sons do jogo.

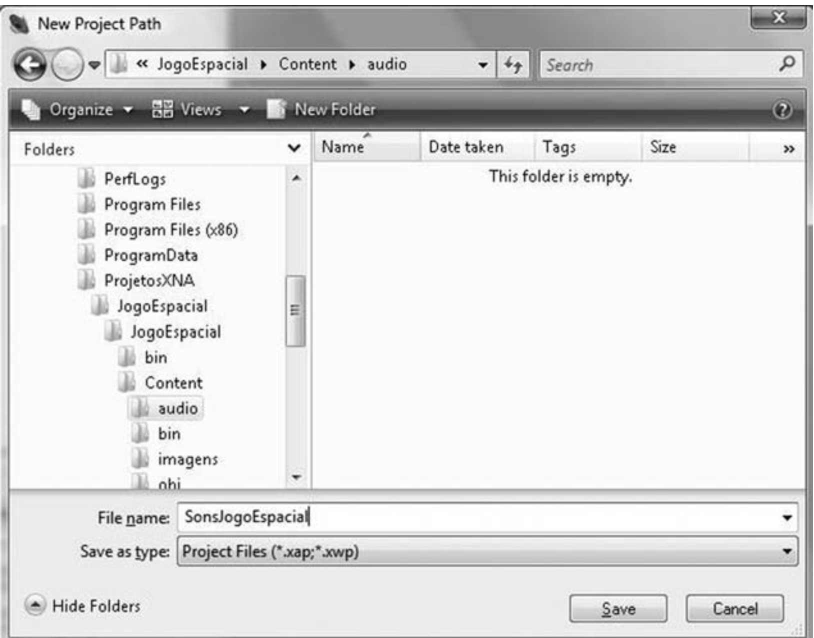


Figura 8.4: Criando a pasta **audio** e nomeando o projeto de sons.
Fonte: XACT.

Para nomear seu projeto de sons, escreva dentro do campo **File name** o texto: **SonsJogoEspacial**. A **Figura 8.4** ilustra o procedimento. Em seguida, clique no botão **Save** para criar seu novo projeto de sons.

Atividade prática 1

Seguindo os procedimentos descritos, abra a ferramenta **XACT** e crie um projeto de sons para o seu jogo. É importante que esse projeto de sons seja criado dentro da pasta **C:\ProjetosXNA\JogoEspacial\JogoEspacial\Content\audio** de seu projeto de jogo.

Acrescentando sons ao seu projeto de sons

Agora já temos o projeto de sons, mas ele está vazio. Para adicionar os sons ao nosso projeto de jogo, precisamos criar um banco de ondas (**Wave Bank**) e um banco de sons (**Sound Bank**) dentro do projeto. Para tal, siga estes passos:

Passo 1: Clique com o botão direito no item **Wave Banks**, localizado no menu esquerdo do seu projeto de sons, e selecione a opção **New Wave Bank**. Esse procedimento pode ser visto na **Figura 8.5**:

Figura 8.5: Acrescentando um banco de ondas ao projeto de sons.
Fonte: XACT.

Repare que foi criada uma janela no lado direito da ferramenta chamada **Wave Bank**. Essa janela corresponde ao seu recém-criado banco de ondas.

Passo 2: Clique com o botão direito no item **Sound Banks**, localizado no menu esquerdo do seu projeto de sons, e selecione a opção **New Sound Bank**. Esse procedimento pode ser visto na **Figura 8.6**:

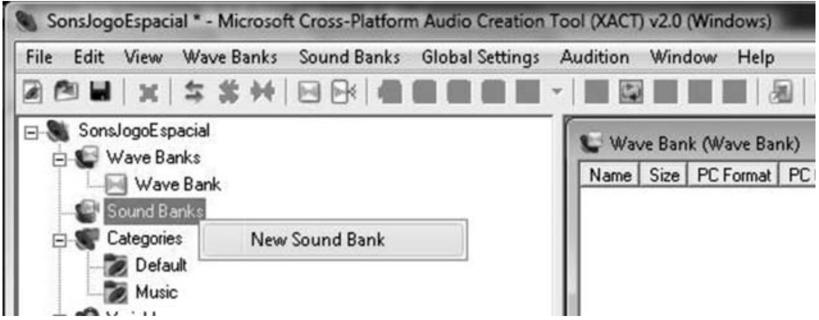


Figura 8.6: Acrescentando um banco de sons ao projeto de sons.
Fonte: XACT.

Repare que apareceu uma nova janela no lado direito da ferramenta chamada **Sound Bank**. Essa janela corresponde a seu recém-criado banco de sons.

Passo 3: Selecione o item **Wave Bank** no menu esquerdo do seu projeto de sons, localizado logo abaixo do item **Wave Banks**. Perceba que a janela do seu banco de ondas ficará em evidência, como na **Figura 8.7:**

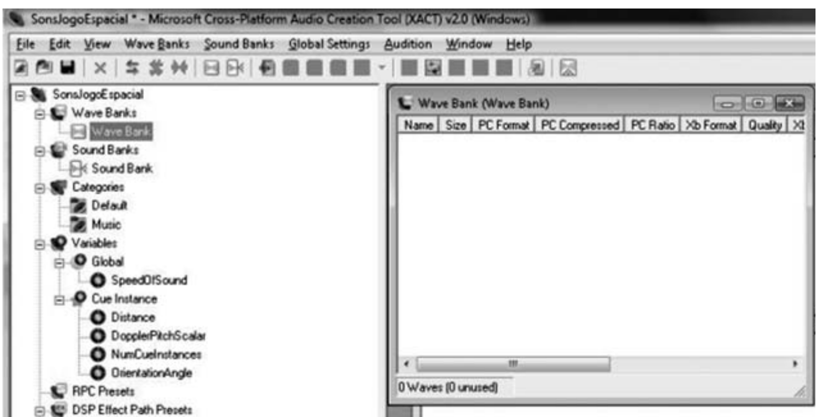


Figura 8.7: Selecionando o banco de ondas.
Fonte: XACT.

Passo 4: Precisamos agora acrescentar dois arquivos de som ao nosso banco de ondas (**Wave Bank**). O primeiro será a trilha sonora

do jogo; o segundo será o som do tiro disparado pela espaçonave do jogador.

Atenção: A ferramenta **XACT** só trabalha com arquivos de som do tipo WAV (**Wave Audio File**).

Procure nas pastas do seu computador esses dois arquivos e copie-os para a pasta **audio** do seu projeto de jogo. Em seguida, abra a pasta **audio** e arraste e solte cada um deles dentro da janela do banco de ondas, para adicioná-los ao seu projeto de sons.

Repare que aparecerão agora dois arquivos em vermelho dentro da janela do banco de ondas (**Wave Bank**), como mostra a **Figura 8.8**. Os dois arquivos escolhidos foram **musica.wav** e **tiro.wav**.

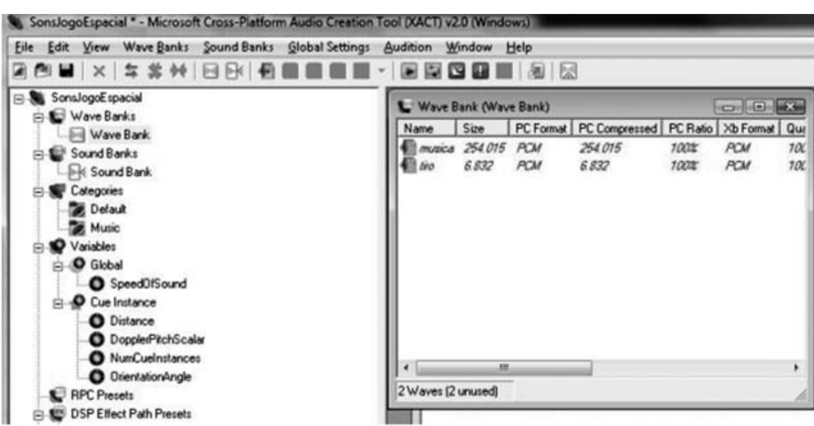


Figura 8.8: Acrescentando arquivos de sons do tipo WAV ao banco de ondas.
Fonte: XACT.

Passo 5: Selecione agora o seu banco de sons (**Sound Bank**), localizado no menu esquerdo do seu projeto de sons, para colocar em evidência a janela do banco de sons. Arraste a janela um pouco para baixo, de forma que você possa visualizar as duas janelas (banco de sons e banco de ondas) ao mesmo tempo, conforme a **Figura 8.9**:

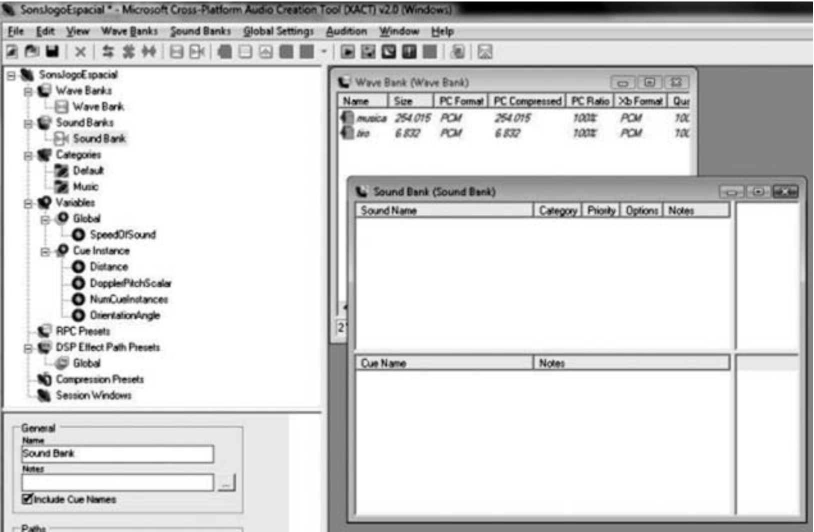


Figura 8.9: Selecionando e posicionando a janela do banco de sons.
Fonte: XACT.

Passo 6: Selecione cada um dos arquivos da janela do seu banco de ondas (**Wave Bank**) e arraste-os para dentro da janela do seu banco de sons (**Sound Bank**), dentro da **área inferior da janela, abaixo de onde aparece o texto Cue Name.**

Observe na **Figura 8.10** que os arquivos do seu banco de ondas agora também estão listados dentro do seu banco de sons.

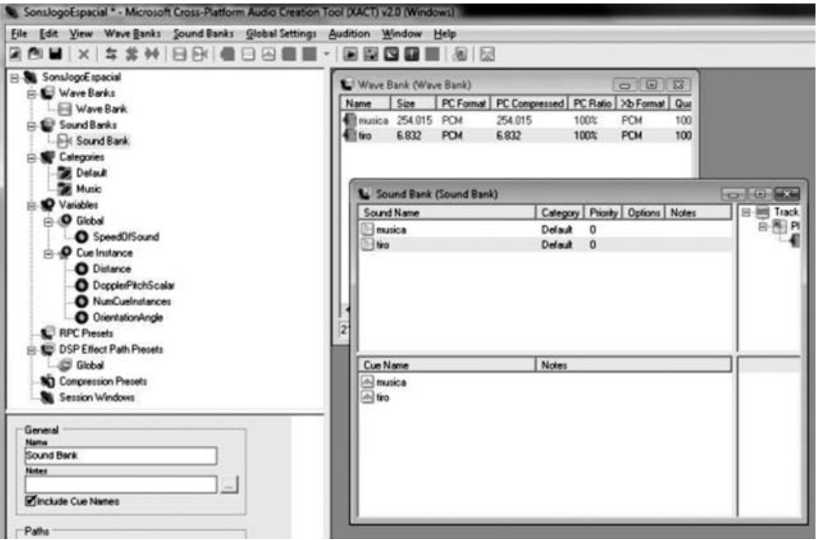
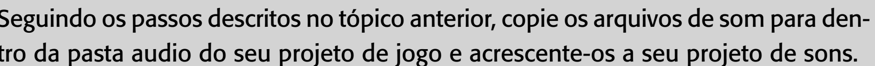


Figura 8.10: Arrastando arquivos do seu banco de ondas para seu banco de sons.
Fonte: XACT.

Passo 7: Pronto! Agora o seu projeto de sons já está completo. Basta salvá-lo utilizando a opção **File\Save Project** do menu superior da ferramenta **XACT**, ou apertando as teclas **Control + S**.

Atividade prática 2

Seguindo os passos descritos no tópico anterior, copie os arquivos de som para dentro da pasta audio do seu projeto de jogo e acrescente-os a seu projeto de sons.



Acrescentando o projeto de sons ao seu projeto de jogo

Parabéns! Agora você já tem o seu projeto de sons criado e configurado com a ferramenta **XACT**. Vamos então acrescentá-lo ao projeto de jogo:

- Abra uma janela no Windows Explorer e procure pela pasta **audio** do seu projeto de jogo.
- Abra também seu projeto de jogo na ferramenta **Visual C#**, de forma que você possa visualizar as janelas da pasta **audio** e do projeto de jogo na tela ao mesmo tempo.
- Arraste sua pasta **audio** para dentro da pasta **Content**, localizada no **Solution Explorer**, que aparece no canto direito da janela do seu projeto de jogo.

Após seguir esses procedimentos, você deverá visualizar uma janela de projeto de jogo semelhante à **Figura 8.11**. Veja que todo o conteúdo que já existia dentro da pasta **audio**, incluindo a própria pasta e os arquivos do projeto de sons, foi acrescentado ao seu projeto de jogo.

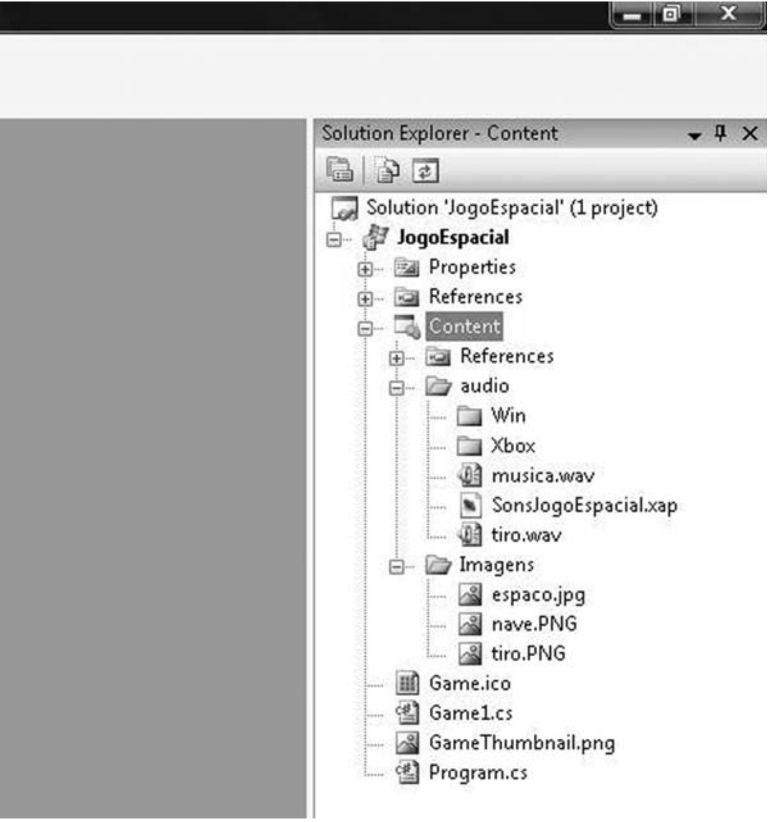


Figura 8.11: Acrescentando a pasta audio ao projeto de jogo.
Fonte: Visual C#.

Utilizando os sons dentro do projeto de jogo

Para utilizar os sons dentro do projeto do jogo, precisaremos realizar alguns ajustes no código do projeto. Para isso, execute os seguintes passos:

Passo 1: Acrescentar três novos atributos ao projeto de jogo, para que ele possa carregar o projeto de sons, o banco de ondas e o banco de sons que acabamos de criar com a ferramenta **XACT**. São eles:

- **Audio Engine –** o projeto de sons.
- **Wave Bank –** o banco de ondas.
- **Audio Bank –** o banco de sons.

Veja como ficou o código do projeto de jogo e acrescente os atributos a seu projeto de jogo.

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

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

    // Definindo um leitor de teclas do teclado
    KeyboardState teclado;

    // Criando uma lista de tiros disparados no jogo
    List<Tiro> TirosDisparados;

    // Atributos do cenário de fundo do jogo
    Texture2D cenario_fundo;
    int posicaoy_cenario_fundo;

    // Atributo para controlar o disparo dos tiros
    bool apertou_tiro;

   // Atributos do projeto de sons
  AudioEngine audioEngine;
  WaveBank waveBank;
  SoundBank soundBank;
```

Passo 2: No método de carga do conteúdo do jogo (**LoadContent**), acrescente três chamadas para carregar os arquivos do projeto de sons, do banco de ondas e do banco de sons no seu projeto de jogo. Eles são os atributos que você acabou de especificar. Repare no código:

```
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
     NaveJogador.CarregarTextura(graphics.
     GraphicsDevice, "../../../Content/
     Imagens/nave.PNG");

    // Carregando o cenário de fundo do jogo
     cenario_fundo = Texture2D.FromFile
     (graphics.GraphicsDevice, "../../../
     Content/Imagens/espaco.jpg");

   // Carregando os arquivos de som do jogo
    audioEngine = new AudioEngine(@"Content\audio\
    SonsJogoEspacial.xgs");
    waveBank = new WaveBank(audioEngine, @"Content\
    audio\Wave Bank.xwb");
    soundBank = new SoundBank(audioEngine,
    @"Content\audio\Sound Bank.xsb");
}
```

Observe que os arquivos carregados estão com extensões diferentes aqui: **xgs** para o projeto de sons e **xwb** para os bancos de ondas e sons. Isso acontece porque a estrutura do projeto do XNA compila automaticamente os arquivos originais, gerando esses arquivos diferentes quando o projeto de jogo é executado.

Passo 3: Acrescente ao início do método de atualização do jogo (**Update**) uma instrução para atualizar o dispositivo de áudio (**AudioEngine**):

```
protected override void Update(GameTime
gameTime)
{
// Atualizando o dispositivo de audio do jogo
audioEngine.Update();
```

Isso fará com que o dispositivo de áudio (**AudioEngine**) esteja sempre ativo dentro do jogo, permitindo a execução dos sons.

Passo 4: Adicione as chamadas para os arquivos de áudio que você colocou no banco de sons. Isso é feito por meio do comando **soundBank.PlayCue**.

Para a trilha sonora, que permanecerá tocando durante todo o jogo, coloque a chamada dentro do método de carga de conteúdo do jogo (**LoadContent**), após a carga dos arquivos de som do jogo:

```
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
     NaveJogador.CarregarTextura(graphics.
     GraphicsDevice, "../../../Content/Imagens/
     nave.PNG");

    // Carregando o cenário de fundo do jogo
     cenario_fundo = Texture2D.FromFile
     (graphics.GraphicsDevice, "../../../Content/
     Imagens/espaco.jpg");

   // Carregando os arquivos de som do jogo
     audioEngine = new AudioEngine(@"Content\
     audio\SonsJogoEspacial.xgs");
```

```
        waveBank = new WaveBank(audioEngine,
        @"Content\audio\Wave Bank.xwb");
        soundBank = new SoundBank(audioEngine,
        @"Content\audio\Sound Bank.xsb");

          // Carregando a música do jogo
          soundBank.PlayCue("musica");
    }
```

Para o som do tiro, coloque a chamada do arquivo de áudio dentro do método de atualização do jogo, logo após a geração do tiro:

```
protected override void Update(GameTime
gameTime)
{
    // Atualizando o dispositivo de audio do
    jogo
    audioEngine.Update();

    // Allows the game to exit
     if (GamePad.GetState(PlayerIndex.One).
     Buttons.Back == ButtonState.Pressed)
     this.Exit();

    // TODO: Add your update logic here
    teclado = Keyboard.GetState();
    if (teclado.IsKeyDown(Keys.Up))
    {
     NaveJogador.Mover(1,this.Window.
     ClientBounds.Width,this.Window.
     ClientBounds.Height);
    }
    if (teclado.IsKeyDown(Keys.Right))
    {
     NaveJogador.Mover(2,this.Window.
     ClientBounds.Width, this.Window.
     ClientBounds.Height);
```

```
}
if (teclado.IsKeyDown(Keys.Down))
{
 NaveJogador.Mover(3, this.Window.
 ClientBounds.Width, this.Window.
 ClientBounds.Height);
}
if (teclado.IsKeyDown(Keys.Left))
{
 NaveJogador.Mover(4, this.Window.
 ClientBounds.Width, this.Window.
 ClientBounds.Height);
}
 if (teclado.IsKeyDown(Keys.Space) &&
 !apertou_tiro) //Apertou o tiro
{
 NaveJogador.Atirar(graphics.
 GraphicsDevice, "../../../Content/
 Imagens/tiro.png", TirosDisparados);
 apertou_tiro = true;

// Carregando o som do tiro
soundBank.PlayCue("tiro");
 }
if (teclado.IsKeyUp(Keys.Space) &&
 apertou_tiro)
 {
     apertou_tiro = false;
 }
```

Passo 5: Pronto! Agora basta executar seu projeto de jogo e ver o resultado.

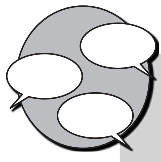
Atividade prática **3**

Seguindo os passos descritos nos tópicos "Acrescentando o projeto de sons ao seu projeto de jogo" e "Utilizando os sons dentro do projeto de jogo", abra seu projeto de jogo espacial na ferramenta **Visual C#** e incorpore a ele o projeto de sons que você criou. Faça tocar a trilha sonora do jogo e quando algum tiro for disparado pela espaçonave do jogador, o som dele também deverá ser tocado.

Atividade prática **4**

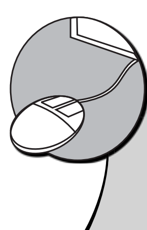
Após ter completado a Atividade Prática 3 e executado o jogo, repare que a trilha sonora toca apenas uma única vez. Altere seu projeto de sons do jogo utilizando a ferramenta **XACT** e configure o arquivo da trilha sonora do jogo para que ele toque indefinidamente, reiniciando a trilha sonora sempre que a música chegar ao fim.

Dica: Utilize apenas a ferramenta **XACT** para alterar seu projeto de sons. O ajuste deve ser feito clicando no arquivo de som da trilha sonora localizada dentro da janela de banco de sons (**SoundBank**).



INFORMAÇÃO SOBRE FÓRUM

Você teve alguma dificuldade para acrescentar os sons ao seu jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.

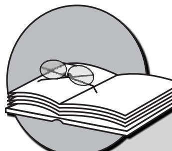


Atividade *online*

Agora que você já está com seu código de projeto do jogo ajustado para tocar os sons, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- Para acrescentar sons a seu projeto de jogo, você precisa utilizar a ferramenta **XACT** (Microsoft **Cross**-Plataform **A**udio **C**reation **T**ool). Ela já está incluída no pacote de instalação do XNA Game Studio.
- Inicie a ferramenta **XACT** clicando no botão **Iniciar** do Windows e selecione no menu a opção **Programas\Xna Game Studio\Tools\Microsoft Cross-Plataform Audio Creation Tool** (**XACT**). Para criar um novo projeto de sons, basta selecionar a opção **File\New Project** e definir como local de salvamento para o projeto a pasta **Content/audio** do seu projeto de jogo.
- Agora você precisa criar um novo banco de ondas e um banco de sons no seu projeto de sons. Para tal, clique com o botão direito no item **Wave Bank** do menu esquerdo do **XACT** e selecione a opção **New Wave Bank**. Faça o mesmo para o banco de sons.
- O **XACT** suporta apenas arquivos de som do tipo **WAV (Wave Audio File)**. Para acrescentar arquivos de sons ao seu projeto de sons, mantenha aberta a janela do banco de ondas (**Wave Bank**) e arraste os arquivos desejados para dentro dela. Em seguida, abra a janela do banco de sons (**Sound Bank**) e arraste para o canto inferior dela (abaixo de **Cue Name**) os arquivos que estão dentro da janela do banco de ondas (**Wave Bank**). Ao final, salve o projeto de jogo utilizando a opção **File\Save Project** do menu superior.
- Você ainda precisa incluir o projeto de sons dentro do seu projeto de jogo. Arraste a pasta **audio** na qual você colocou todos os seus arquivos de som e o projeto de sons para dentro da pasta **Content** no **Solution Explorer** do seu projeto de jogo.
- Para poder tocar os sons dentro do jogo, será necessário ajustar o código do projeto de jogo criando três atributos novos: **AudioEngine**, **WaveBank** e **SoundBank**. Além disso, você precisa alterar também o método de carga de conteúdo (**LoadContent**), para que seja feita a carga dos arquivos correspondentes do seu projeto de sons dentro do projeto de jogo. Por fim, acrescente a chamada **audioEngine.Update** ao método de atualização do jogo (**Update**) e utilize o comando **soundBank.PlayCue** para tocar os sons.



Informação sobre a próxima aula

Na próxima aula, você verá como verificar colisões de objetos dentro do jogo.

Tratando colisões de objetos no jogo

Meta da aula

Fazer desaparecer da tela do jogo a espaçonave inimiga quando um tiro acertá-la.

objetivo

Ao final desta aula, você deverá ser capaz de:

1. verificar se um tiro acertou uma espaçonave do jogo.

Pré-requisitos

Estar familiarizado com a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na Aula 3; ter seu projeto de jogo atualizado conforme o conteúdo da Aula 7, que inclui a produção de tiros pela espaçonave.

INTRODUÇÃO

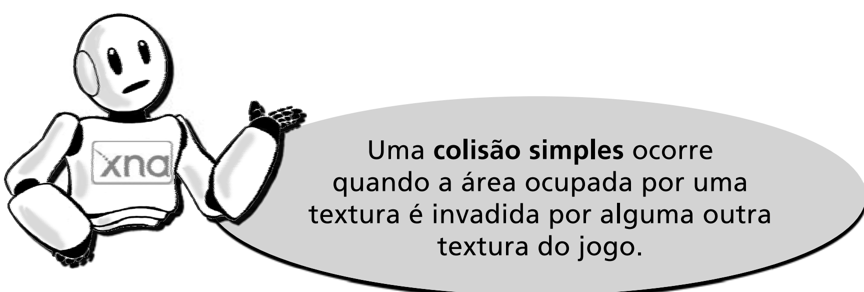
Na última aula, incorporamos diversos sons aos acontecimentos do nosso jogo.

Figuras 9.1 e 9.2: Exemplos de colisão.

Você se lembra da lei da Física que diz que dois objetos não podem ocupar o mesmo lugar no espaço ao mesmo tempo?
Exatamente! Para trazer um pouco mais de realismo ao jogo, agora você vai aprender como tratar as colisões entre os objetos do jogo.

VERIFICANDO COLISÕES SIMPLES ENTRE OS OBJETOS DO JOGO

Veja agora como funciona um teste de colisão simples.



A **Figura 9.3** mostra um exemplo de **colisão simples** entre duas texturas dentro do jogo:

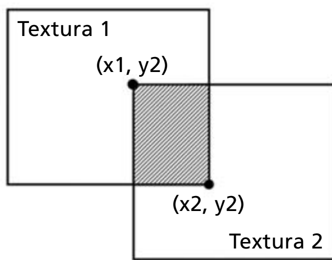


Figura 9.3: Colisão simples entre duas texturas.
Fonte: Word.

A verificação de uma **colisão simples** no jogo é muito fácil. Repare que as texturas das naves e dos tiros são todas retangulares. Precisamos apenas testar se o retângulo correspondente à primeira textura intercepta o retângulo referente a uma segunda textura.

Vamos criar um método chamado **ColisaoSimples** para realizar essa verificação. O método precisará receber como parâmetros as duas texturas a serem verificadas e as posições em que elas estão localizadas na tela do jogo.

```
public bool ColisaoSimples(Texture2D Textura1, Texture2D
Textura2, int PosicaoX1, int PosicaoY1, int PosicaoX2,
int PosicaoY2)
{
     return (PosicaoX1 + Textura1.Width > PosicaoX2 &&
             PosicaoX1 < PosicaoX2 + Textura2.Width &&
             PosicaoY1 + Textura1.Height > PosicaoY2 &&
         PosicaoY1 < PosicaoY2 + Textura2.Height);
}
```

Observe que o método **ColisaoSimples** retorna um resultado lógico (verdadeiro ou falso), indicando se houve ou não a colisão entre as texturas. O método realiza quatro verificações; se todas forem verdadeiras, a colisão terá ocorrido. As verificações são:

- A posição x da textura 1 mais a largura da textura 1 ultrapassam a posição x da textura 2?
- A posição x da textura 2 mais a largura da textura 2 ultrapassam a posição x da textura 1?
- A posição y da textura 1 mais a altura da textura 1 ultrapassam a posição y da textura 2?
- A posição y da textura 2 mais a altura da textura 2 ultrapassam a posição y da textura 1?

Para testar as colisões, precisamos criar também uma espaçonave inimiga dentro do nosso jogo e fazer alguns ajustes:

Passo 1: Acrescente o atributo **NaveInimiga** aos atributos do jogo:

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

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

    // Definindo a espaçonave inimiga no jogo
   Espaconave NaveInimiga;
```

Passo 2: Precisamos acrescentar à classe Espaçonave alguns métodos para configurar as posições x e y da espaçonave e retornar à textura:

```
 public int PosicaoX()
{
    return _posicaox;
}

public int PosicaoY()
{
    return _posicaoy;
}

public void PosicaoX(int p_posicaox)
{
    _posicaox = p_posicaox;
}

public void PosicaoY(int p_posicaoy)
{
    _posicaoy = p_posicaoy;
}

public Texture2D Textura()
{
    return _textura;
}
```

Passo 3: Assim como a espaçonave do jogador, a nave inimiga também possui nome e posição inicial. Configure a espaçonave do jogador e a inimiga no método de inicialização do jogo (**Initialize**):

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

      // Configurado a nave do jogador
      NaveJogador.Nome("Falcão Justiceiro");
      NaveJogador.PosicaoX(300);
      NaveJogador.PosicaoY(300);

      // Criando efetivamente a espaçonave inimiga
      NaveInimiga = new Espaconave();

      // Configurando a nave inimiga
      NaveInimiga.Nome("Soldado do Espaço");
      NaveInimiga.PosicaoX(200);
      NaveInimiga.PosicaoY(10);
```

Passo 4: A nave inimiga também precisa aparecer na tela do jogo. Carregue uma textura para ela dentro do método de carga de conteúdo (**LoadContent**) do jogo e a desenhe no método de exibição do jogo (**Draw**):

```
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
      NaveJogador.CarregarTextura(graphics.GraphicsDevice,
      "../../../Content/Imagens/nave.PNG");
      NaveInimiga.CarregarTextura(graphics.GraphicsDevice,
      "../../../Content/Imagens/inimigo.PNG");
```

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

           // Desenhando o cenário de fundo do jogo
           spriteBatch.Begin();
      spriteBatch.Draw(cenario_fundo, new Rectangle(0,
      posicaoy_cenario_fundo, this.Window.ClientBounds.
      Width, this.Window.ClientBounds.Height), Color.
      White);
      spriteBatch.Draw(cenario_fundo, new Rectangle(0,
      posicaoy_cenario_fundo - this.Window.ClientBounds.
      Height, this.Window.ClientBounds.Width, this.
      Window.ClientBounds.Height), Color.White);
           spriteBatch.End();

           // TODO: Add your drawing code here
           NaveJogador.Desenhar(spriteBatch);
           NaveInimiga.Desenhar(spriteBatch);
```

Lembre-se de adicionar a imagem da nave inimiga à pasta Imagens, localizada dentro da pasta **Content** do seu projeto do jogo. Clique com o botão direito sobre a pasta Imagens dentro do **Solution Explorer** e utilize a opção **Add/Existing Item** para acrescentá-la ao seu projeto de jogo.

Passo 5: Vamos agora testar a colisão entre os tiros e a espaçonave inimiga. Ajuste o método de atualização do jogo (**Update**) da seguinte forma após a movimentação dos tiros:

```
 // Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados.
 GetRange(0,TirosDisparados.Count))
{
oTiro.Mover();
```

```
 // Testando a colisão do tiro com o inimigo
if  (ColisaoSimples(oTiro.Textura(), NaveInimiga.
Textura(), oTiro.PosicaoX(), oTiro.PosicaoY(),
NaveInimiga.PosicaoX(), NaveInimiga.PosicaoY()))
        {
             // Removendo o tiro
             TirosDisparados.Remove(oTiro);

             // Removendo a espaçonave inimiga da
             tela
             NaveInimiga.PosicaoX(-100);
             NaveInimiga.PosicaoY(-100);
        }
```

Repare que é utilizado um artifício para fazer a espaçonave inimiga desaparecer da tela do jogo. As posições x e y da nave inimiga são atualizadas para o valor -100, localizado fora da tela do jogo.

Excelente! Agora você já é capaz de verificar todas as colisões de objetos dentro do seu projeto de jogo. Execute o jogo e veja o resultado.

Atividade prática 1

Procure uma imagem para a espaçonave inimiga. Seguindo os procedimentos descritos anteriormente, abra seu projeto de jogo na ferramenta Visual C# e acrescente ao seu jogo a espaçonave inimiga e uma verificação de **colisões simples** entre os tiros disparados e a espaçonave inimiga, fazendo a nave inimiga e o tiro desaparecerem da tela do jogo após a colisão.

VERIFICANDO COLISÕES AVANÇADAS ENTRE OS OBJETOS DO JOGO

Você vai agora aprender a verificar colisões de forma mais detalhada.

O método **ColisaoSimples** resolve a maioria das colisões existentes. Contudo, algumas texturas possuem uma parte transparente. Para esses casos, precisaremos utilizar um teste mais avançado de colisão das texturas.

Para o teste avançado de colisão de texturas, vamos verificar:

• se as áreas das texturas estão sobrepostas – **ColisaoSimples**;

• caso estejam, se dentro da área sobreposta existe algum ponto não transparente da textura 1 sobrepondo-se outro ponto não transparente da textura 2.

Veja como implementamos o teste avançado de colisões:

```
public bool ColisaoAvancada(Texture2D Textura1,Texture2D
Textura2, int PosicaoX1, int PosicaoY1, int PosicaoX2,
int PosicaoY2)
{
            if (ColisaoSimples(Textura1, Textura2,
            PosicaoX1, PosicaoY1, PosicaoX2, PosicaoY2))
            {
                // Obtendo os bits das texturas
                uint[] BitsTextura1, BitsTextura2;
                BitsTextura1 = new uint[Textura1.Width *
                Textura1.Height];
                BitsTextura2 = new uint[Textura2.Width *
                Textura2.Height];

                Textura1.GetData<uint>(BitsTextura1);
                Textura2.GetData<uint>(BitsTextura2);

                // Obtendo as coordenadas x e y mínima e
                máxima
                // de intersecção entre as texturas
                int x1 = Math.Max(PosicaoX1,
                PosicaoX2);
```

```
        int y1 = Math.Max(PosicaoY1, PosicaoY2);

        int  x2  =  Math.Min(PosicaoX1  +  Textura1.
        Width,PosicaoX2 + Textura2.Width);
    int y2 = Math.Min(PosicaoY1 + Textura1.Height,PosicaoY2
+ Textura2.Height);

        // Percorrendo a área de intersecção das Naves
        // para verificar se os pixels são transparentes
        ou não
        for (int linha = y1; linha < y2; linha++)
         {
        for (int coluna = x1; coluna < x2; coluna++)
         {
       // Se ambos os pixels das naves A e B verificados
       // tem cor então = Colisão
      if (   ((BitsTextura1[(coluna - PosicaoX1) + (linha
      - PosicaoY1) * Textura1.Width]
      & 0xFF000000) >> 24) > 20 &&
     ((BitsTextura2[(coluna - PosicaoX2) + (linha -
      PosicaoY2) * Textura2.Width]
     & 0xFF000000) >> 24) > 20)
                    {
                        return true;
                    }
                }
            }
        }
        return false;
}
```

Repare que a primeira verificação realizada dentro do método **ColisaoAvancada** é chamar o método **ColisaoSimples**, para verificar há alguma sobreposição das texturas.

Caso haja, é utilizado o método **GetData** da classe de texturas, para extrair as cores dos pontos existentes nas duas texturas. Essas cores são armazenadas dentro de duas listas, que são **BitsTextura1** e **BitsTextura2**.

Por fim, toda a área sobreposta entre as duas texturas é percorrida utilizando dois comandos de repetição (**for**) encadeados. Dentro deles é realizado um teste para ver se em alguma parte da área sobreposta ambos os pontos são coloridos, ou seja, se o valor da cor de uma posição na lista **BitsTextura1** é maior que 20 e se o valor da cor nessa mesma posição na lista **BitsTextura2** também é maior que 20. Isso configura a colisão avançada de texturas.

Vamos agora modificar nosso teste de colisão entre os tiros disparados e a espaçonave inimiga no método de atualização do jogo (**Update**), trocando a chamada **ColisaoSimples** por **ColisaoAvancada:**

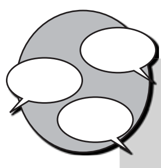
```
 // Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados.
GetRange(0,TirosDisparados.Count))
       {
           oTiro.Mover();

// Testando a colisão do tiro com a nave inimiga
if (ColisaoAvancada(oTiro.Textura(), NaveInimiga.
 Textura(), oTiro.PosicaoX(), oTiro.PosicaoY(),
  NaveInimiga.PosicaoX(), NaveInimiga.PosicaoY()))
           {
// Removendo o tiro
TirosDisparados.Remove(oTiro);
// Removendo a espaçonave inimiga da tela do jogo
               NaveInimiga.PosicaoX(-100);
               NaveInimiga.PosicaoY(-100);
           }
```

Atividade prática 2

Seguindo os procedimentos descritos, abra seu projeto de jogo na ferramenta Visual C# e acrescente a seu jogo a verificação de **colisões avançadas** entre os tiros e a espaçonave inimiga, fazendo a nave inimiga e o tiro desaparecerem da tela do jogo após a colisão.

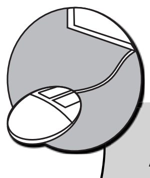
Atividade prática 3

Agora que você já sabe verificar as colisões, programe dentro do seu projeto de jogo um teste de **colisão avançada** entre a espaçonave do jogador e a espaçonave inimiga.



INFORMAÇÃO SOBRE FÓRUM

Você teve alguma dificuldade para verificar as colisões dentro do jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.

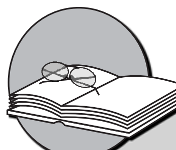


Atividade *online*

Agora que você já está com seu código de projeto do jogo ajustado para verificar colisões entre objetos do jogo, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- Uma **colisão simples** ocorre quando a área ocupada por uma textura é invadida por alguma outra textura do jogo.
- Para verificar colisões entre objetos no seu projeto de jogo, você precisa testar se o retângulo correspondente à primeira textura intercepta o retângulo referente a uma segunda textura.
- É necessário acrescentar uma espaçonave inimiga ao jogo para realizar essas verificações. Isso implica:
    - o acrescentar o atributo **NaveInimiga** ao código de criação do jogo;
    - o configurar o atributo **NaveInimiga** dentro do método de inicialização do jogo (**Initialize**);
    - o acrescentar os métodos **PosicaoX**, **PosicaoY** e **Textura** à classe Espaconave para configurar a posição das espaçonaves e obter a textura da espaçonave;
    - o carregar a textura da espaçonave inimiga no método de carga de conteúdo do jogo (**LoadContent**);
    - o desenhar a espaçonave inimiga no método de exibição do jogo (**Draw**).
- É necessário também acrescentar o teste de colisão simples entre o tiro disparado e a espaçonave inimiga no método de atualização do jogo (**Update**).
- Para o teste avançado de colisão de texturas, verifica-se:
    - o se as áreas das texturas estão sobrepostas – **ColisaoSimples**;
    - o caso estejam, se dentro da área sobreposta existe algum ponto não transparente da textura 1 sobrepondo outro ponto não transparente da textura 2.



Informação sobre a próxima aula

Na próxima aula, você verá como produzir efeitos duradouros, como explosões, dentro do jogo.

Produzindo efeitos duradouros (explosões) no jogo

Meta da aula

Fazer explodir a espaçonave inimiga quando um tiro acertá-la.

objetivo

Ao final desta aula, você deverá ser capaz de:

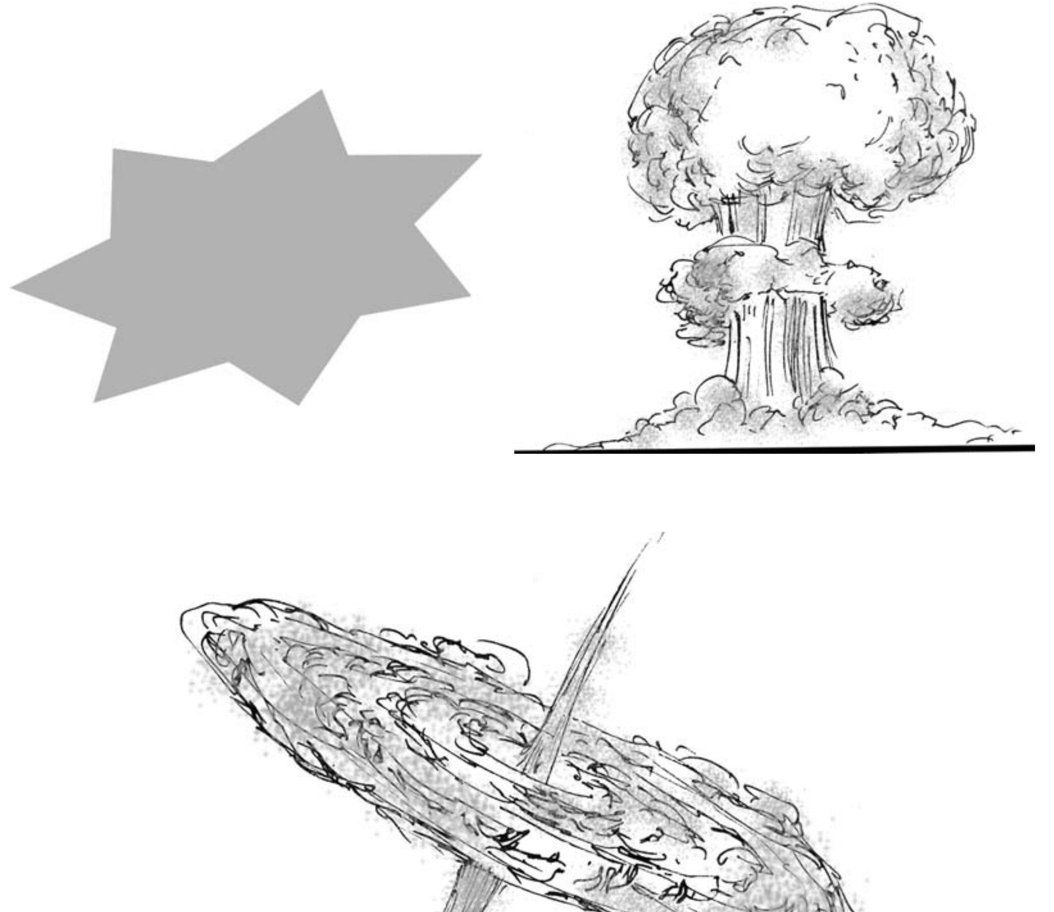
gerar um efeito duradouro de explosão no jogo.

Pré-requisitos

Estar familiarizado com a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na Aula 3; ter seu projeto de jogo atualizado conforme o conteúdo das Aulas 8 e 9, que incluem a geração de sons para os eventos do jogo e a verificação de colisões entre os tiros disparados e a espaçonave inimiga.

INTRODUÇÃO

Na última aula, acrescentamos uma espaçonave inimiga ao jogo e incorporamos verificações de colisão entre os tiros disparados e a nave inimiga.

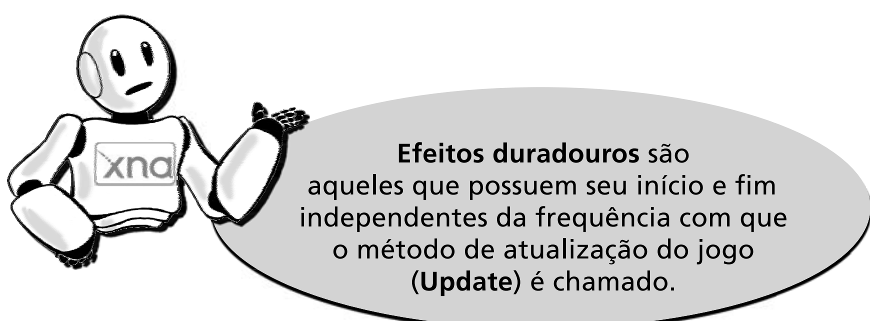


Figuras 10.1, 10.2 e 10.3: Exemplos de explosão.

Agora vamos trazer um pouco mais de emoção ao jogo! Você aprenderá como produzir explosões e outros efeitos duradouros.

PRODUZINDO EXPLOSÕES DENTRO DO JOGO

A explosão é um efeito duradouro. Como assim?



Vamos criar uma explosão:

Passo 1: Crie um atributo no jogo para controlar o tempo de duração da explosão e outro para indicar se ela ocorreu.

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

```
// Definindo o controle de tempo da explosão
double tempoExplosao;
bool explodiu;
```

Repare que o tipo do atributo **tempoExplosao** é duplo (**double**). Ele consegue armazenar um número com o dobro do tamanho de um atributo do tipo inteiro (**int**). Essa capacidade é importante, pois estamos lidando com medição do tempo, o que requer maior precisão.

Passo 2: Inicialize os atributos **tempoExplosao** e **explodiu** com os valores zero (0) e falso no método de inicialização do jogo (**Initialize**).

```
protected override void Initialize()
{
    // Inicializando o controle da explosão
    tempoExplosao = 0;
    explodiu = false;
```

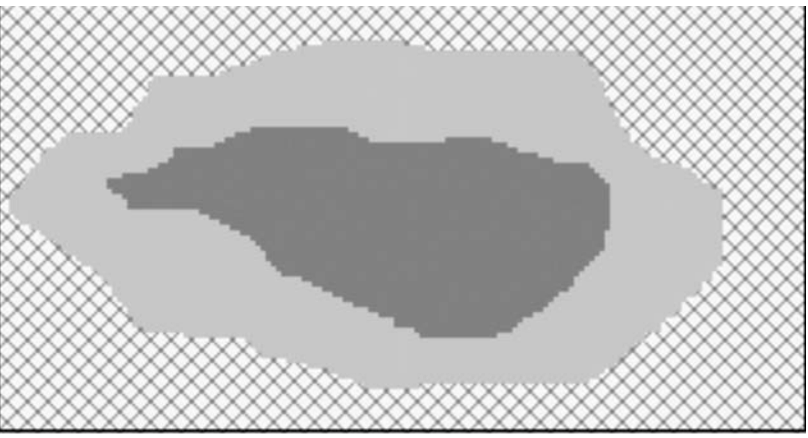


Figura 10.4: Exemplo de textura utilizada para o efeito de explosão.
Fonte: Paintbrush.

Passo 3: Altere o método de atualização do jogo (**Update**), incluindo dentro do teste de colisão o seguinte:

- ajuste do atributo **explodiu** para o valor verdadeiro (**true**);
- uma instrução para carregar a textura da explosão no lugar da textura normal da espaçonave inimiga;

• toque do som da explosão.

Veja como ficou:

```
// Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados.
GetRange(0,TirosDisparados.Count))
     {
         oTiro.Mover();

// Testando a colisão do tiro com a nave inimiga
if (!explodiu && ColisaoAvancada(oTiro.Textura(),
NaveInimiga.Textura(), oTiro.PosicaoX(), oTiro.
PosicaoY(), NaveInimiga.PosicaoX(), NaveInimiga.
PosicaoY()))
         {
 // Removendo o tiro
         TirosDisparados.Remove(oTiro);

         explodiu = true;

 // Carregando a textura de explosão inimiga
NaveInimiga.CarregarTextura(graphics.GraphicsDevice,
"../../../Content/Imagens/explosao.png");

 // Carregando som da explosão
 soundBank.PlayCue("explosao");
    }
```

Repare que foi acrescentado um teste para verificar se o atributo **explodiu** é falso (**!explodiu**). Isso é feito para evitar que a explosão da mesma espaçonave aconteça mais de uma vez.

Será necessário incorporar ao projeto de jogo uma imagem para a explosão, que será carregada no lugar da imagem da espaçonave inimiga quando esta explodir.

Para acrescentar imagens ao seu projeto de jogo, clique com o botão direito em cima da pasta **Imagens**, localizada dentro da pasta **Content** do **Solution Explorer**. Em seguida, selecione a opção **Add/ Existing Item** e escolha a imagem da explosão.

Observe também que o som da explosão ainda não existe dentro do seu projeto de sons do jogo. Será necessário abrir o projeto de sons com a ferramenta **XACT** e acrescentar um som de explosão ao projeto de sons, como descrito na Aula 8, salvando-o. Também será necessário incluir o arquivo de som da explosão dentro da pasta **Audio** do seu projeto de jogo, seguindo o mesmo procedimento descrito acima para a imagem da explosão.

Passo 4: Armazene o tempo decorrido desde que houve a explosão dentro do atributo **TempoExplosao** e verifique se ultrapassou 100 milissegundos, que é a duração da explosão.

Após ultrapassar esse tempo, vamos colocar a nave inimiga em uma posição fora da tela do jogador, criando um efeito visual de desaparecimento da espaçonave após a explosão.

O melhor lugar para colocar esse código é antes da atualização do cenário de fundo do jogo que fica localizada no final do método de atualização do jogo (**Update**).

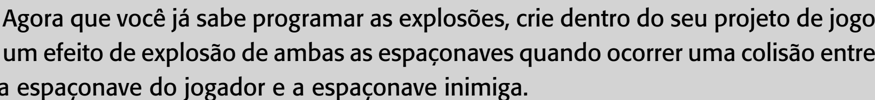
```
// Controle da explosão da nave inimiga
if (explodiu)
{
tempoExplosao += gameTime.ElapsedGameTime.
Milliseconds;
    if (tempoExplosao > 100)
    {
// Fazendo desaparecer a nave inimiga
          NaveInimiga.PosicaoX(-100);
          NaveInimiga.PosicaoY(-100);
          explodiu = false;
          tempoExplosao = 0;
    }
 }

// Movimentando o cenário de fundo do jogo
posicaoy_cenario_fundo += NaveJogador.Velocidade() / 2;
if (posicaoy_cenario_fundo >= this.Window.ClientBounds.
Height)  posicaoy_cenario_fundo = 0;
  }
```

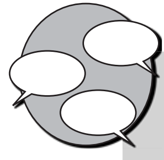
Atividade prática 1

Procure uma imagem para a explosão. Seguindo os quatro passos descritos anteriormente, abra seu projeto de jogo na ferramenta Visual C# e programe uma explosão com duração de 100 milissegundos dentro do seu jogo, quando a espaçonave inimiga for acertada por algum tiro. Faça também a nave inimiga e o tiro desaparecerem da tela após o fim da explosão.

Atividade prática 2

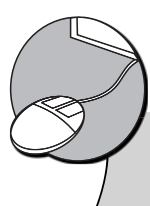
Agora que você já sabe programar as explosões, crie dentro do seu projeto de jogo um efeito de explosão de ambas as espaçonaves quando ocorrer uma colisão entre a espaçonave do jogador e a espaçonave inimiga.





INFORMAÇÃO SOBRE FÓRUM

Você teve alguma dificuldade para criar explosões dentro do jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.

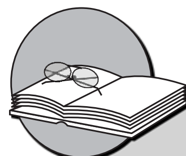


Atividade *online*

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

RESUMO

- Explosão é um efeito duradouro. **Efeitos duradouros** são aqueles que possuem seu início e fim independentes da frequência com que o método de atualização do jogo (**Update**) é chamado.
- Para produzir uma explosão dentro do jogo, será necessário fazer os seguintes ajustes no código do projeto de jogo:
  - o criar e inicializar os atributos **tempoExplosao** (**double**) e **explodiu** (**boolean**), para controlar o efeito da explosão;
  - o acrescentar ao projeto de sons um novo som para a explosão, utilizando a ferramenta **XACT** e salvando o projeto;
  - o acrescentar ao projeto de jogo uma imagem para a explosão. Utilize a opção Add/Existing Item do Solution Explorer para fazê-lo;
  - o acrescentar ao projeto de jogo o arquivo de som da explosão. Utilize a opção Add/Existing Item do Solution Explorer para fazê-lo;
  - o no método de atualização do jogo (**Update**), ao ocorrer a colisão, atualizar o atributo explodiu para verdadeiro, trocar a textura da nave inimiga para a imagem da explosão e tocar o som da explosão;
  - o adicionar um teste ao final do método de atualização do jogo (**Update**) para avaliar se o tempo de explosão ultrapassou 100 milissegundos e remover a espaçonave da tela após esse tempo.



Informação sobre a próxima aula

Na próxima aula, você verá como tornar a espaçonave inimiga mais inteligente dentro do jogo.

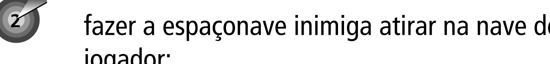
Tornando a espaçonave inimiga mais inteligente

Meta da aula

Tornar a espaçonave inimiga mais ágil e inteligente.

objetivos

Ao final desta aula, você deverá ser capaz de:

1. fazer a espaçonave inimiga tentar escapar dos tiros disparados pelo jogador;

2. fazer a espaçonave inimiga atirar na nave do jogador;

3. fazer a espaçonave inimiga se aproximar da nave do jogador.

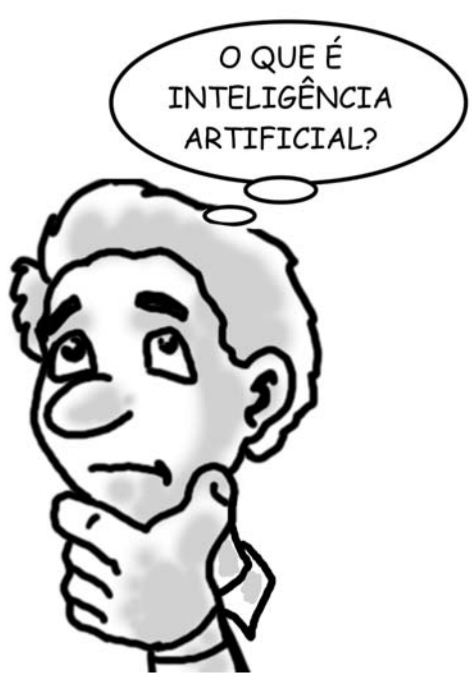
Pré-requisitos

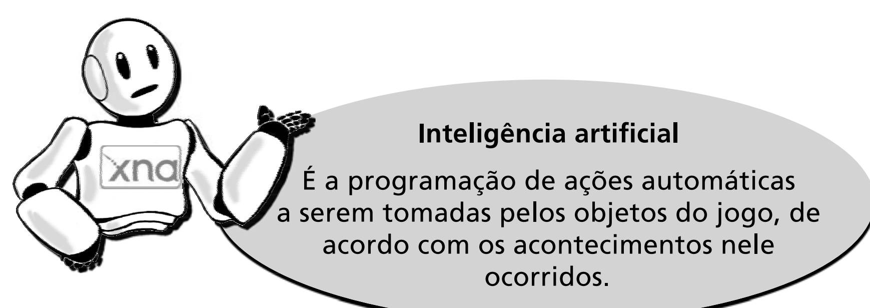
Estar familiarizado com a ferramenta XNA Game Studio, conceito abordado na Aula 2; possuir um computador com as ferramentas Visual C# e XNA Game Studio instaladas, conforme explicado na Aula 3; ter seu projeto de jogo atualizado conforme o conteúdo da Aula 9, que inclui a verificação de colisões entre os tiros disparados e a espaçonave inimiga.

INTRODUÇÃO

Na última aula, você aprendeu a programar efeitos duradouros dentro do jogo, como explosões. Acrescentamos o efeito de explosão quando uma espaçonave inimiga era acertada pelo tiro disparado pela nave do jogador.

Agora, vamos trazer um pouco mais de inteligência ao jogo. Você verá como programar algumas táticas para a espaçonave inimiga se tornar mais desafiadora.





Como podemos tornar a espaçonave inimiga mais inteligente?
Precisamos programar algumas ações automáticas a serem tomadas pela nave inimiga, com base nas ações do jogador durante o jogo.
Quais ações automáticas podem tornar a espaçonave inimiga mais inteligente?
• esquivar-se dos tiros disparados pelo jogador;
• atirar na espaçonave do jogador;
• aproximar-se da espaçonave do jogador.

ESCAPANDO DOS TIROS

Vamos programar a espaçonave inimiga para tentar se esquivar dos tiros disparados pelo jogador.

Passo 1: Adicione o método **Próximo** à classe Espaçonave, para verificar se um objeto está próximo da nave. Esse método é muito importante, e será a base para todos os outros testes que iremos fazer.

```
 // Testa a proximidade em relação a um objeto
public bool Proximo(int p_proximidade, int p_
 posicaox_objeto, int p_posicaoy_objeto)
{
// Testando se o objeto está próximo da espaçonave
if (_posicaox + _textura.Width + p_proximidade >
 p_posicaox_objeto &&
       _posicaox < p_posicaox_objeto + p_
       proximidade &&
       _posicaoy + _textura.Height + p_proximidade
       > p_posicaoy_objeto &&
       _posicaoy < p_posicaoy_objeto + p_
       proximidade)
   {
       return true;
   }
   else return false;
}
```

O teste para verificar se o objeto está próximo da espaçonave é bastante semelhante ao teste de colisão, apenas inclui mais um valor de proximidade para verificar quando o objeto está chegando mais perto da nave.

Passo 2: Acrescente um método **Escapar** à classe Espaçonave, para fazer com que ela se movimente na direção oposta aos tiros disparados no jogo quando esses tiros se aproximarem dela.

```
// Fugir de um tiro
public void Escapar(int p_posicaox_objeto, int
p_posicaoy_objeto, int p_posicaox_maxima, int p_
posicaoy_maxima)
{
// Testando se o objeto está próximo da espaçonave
 if ( Proximo(50, p_posicaox_objeto, p_posicaoy_
 objeto) )
       {
 // Tiro abaixo e à esquerda - mover para cima e
direita
if (p_posicaox_objeto >= _posicaox && p_posicaoy_
objeto > _posicaoy)
            {
Mover(1, p_posicaox_maxima, p_posicaoy_maxima);
Mover(4, p_posicaox_maxima, p_posicaoy_maxima);
            }

// Tiro abaixo e à direita - mover para cima e
esquerda
 if (p_posicaox_objeto < _posicaox && p_posicaoy_
  objeto > _posicaoy)
            {
Mover(1, p_posicaox_maxima, p_posicaoy_maxima);
Mover(2, p_posicaox_maxima, p_posicaoy_maxima);
            }
       }
}
```

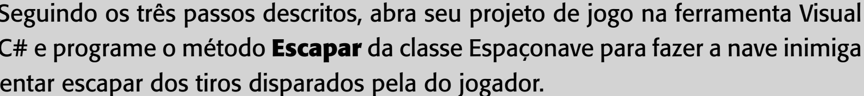
Repare que o método Escapar recebe quatro parâmetros: as posições x e y do objeto do qual a espaçonave está tentando escapar e as posições-limite x e y da tela.

Observe que o movimento da espaçonave é sempre na direção contrária à posição atual do objeto em relação a ela, ou seja, se o objeto está abaixo e à esquerda da nave, por exemplo, esta se movimentará para cima e para a direita.

Passo 3: Acrescente ao método de atualização do jogo (**Update**) uma instrução para chamar o método **Escapar** da espaçonave inimiga. O melhor local para fazer isso é logo após a movimentação de cada tiro. Veja como ficou:

```
// Movendo os tiros do jogo
foreach (Tiro oTiro in TirosDisparados.
GetRange(0,TirosDisparados.Count))
        {
            oTiro.Mover();

 // Inimigo foge dos tiros
 NaveInimiga.Escapar(oTiro.PosicaoX(),
                oTiro.PosicaoY(),
                this.Window.ClientBounds.Width,
                this.Window.ClientBounds.Height);
```

Atividade prática 1



Seguindo os três passos descritos, abra seu projeto de jogo na ferramenta Visual C# e programe o método **Escapar** da classe Espaçonave para fazer a nave inimiga tentar escapar dos tiros disparados pela do jogador.

Atirando contra o jogador

Vamos agora programar a espaçonave inimiga para atacar a nave do jogador atirando contra ela.

Passo 1: Acrescente um método **Atacar** à classe Espaçonave para fazer com que ela se decida se irá atacar, caso algum objeto esteja se aproximando.

```
// Atacar um objeto inimigo
  public bool Atacar(int p_posicaox_objeto, int p_
   posicaoy_objeto)
  {
if (Proximo(200, p_posicaox_objeto, p_posicaoy_
objeto))
        return true;
        else
        return false;
  }
```

Repare que o método **Atacar** é apenas uma chamada ao método **Proximo** com o valor 200, ou seja, apenas decide se a espaçonave deve ou não atacar com base na proximidade do objeto inimigo em relação à espaçonave.

Passo 2: Ajuste o método **Atirar** da classe **Espaçonave** para que receba um novo parâmetro, que é a direção do tiro.

```
public void Atirar(int p_direcao_tiro, GraphicsDevice
p_dispositivo_grafico, string p_local_textura_tiro,
List<Tiro> p_lista_tiros)
```

```
{
   Tiro oTiro;
   // Cria um tiro
      oTiro = new Tiro(_potencia_tiro, p_direcao_tiro,
      _velocidade_tiro, _posicaox, _posicaoy);

      // Carrega a textura do tiro
      oTiro.CarregarTextura(p_dispositivo_grafico,
      p_local_textura_tiro);

      // Ajusta a posição inicial do tiro
      oTiro.PosicaoX(oTiro.PosicaoX() + (_textura.
      Width /2));

   if (p_direcao_tiro == 3) // Tiro para baixo
   {
   oTiro.PosicaoY(oTiro.PosicaoY() + _textura.
   Height);
   }

   // Acrescenta o tiro na lista de tiros do jogo
   p_lista_tiros.Add(oTiro);
}
```

Observe que foi acrescentado também um ajuste para a posição inicial y do tiro caso a direção do tiro seja para baixo (direção 3). Isso foi feito para que ele saia abaixo da textura da espaçonave inimiga, e não por dentro dela.

Passo 3: Acrescente ao método de atualização do jogo (**Update**) uma instrução para chamar o método **Atacar** da espaçonave inimiga. O melhor local para fazer isso é logo após a movimentação da espaçonave do jogador. Veja como ficou:

```
if (teclado.IsKeyUp(Keys.Space) && apertou_tiro)
{
    apertou_tiro = false;
}

// Atirando contra o jogador
if (NaveInimiga.Atacar(NaveJogador.PosicaoX(),
NaveJogador.PosicaoY()))
{
NaveInimiga.Atirar(3,graphics.
GraphicsDevice, "../../../Content/Imagens/tiro.png",
TirosDisparados);
}
```

Atividade prática 2

Seguindo os três passos descritos, abra seu projeto de jogo na ferramenta Visual C# e programe o método **Atacar** da classe Espaçonave para fazer a nave inimiga tentar atirar contra a do jogador.

Aproximando-se do jogador

Vamos agora programar a espaçonave inimiga para se aproximar da nave do jogador, indo na direção dela quando esta chega perto.

Passo 1: Acrescente um método **Aproximar** à classe Espaçonave para fazer com que ela se movimente na direção da nave do jogador quando esta ficar próxima.

```
// Se aproximar do objeto
public void Aproximar(int p_posicaox_objeto, int p_
posicaoy_objeto, int p_posicaox_maxima, int p_
posicaoy_maxima)
  {
// Testando se o objeto está longe da espaçonave
if (Proximo(250, p_posicaox_objeto, p_posicaoy_
objeto))
```

```
{
// Objeto abaixo e à esquerda - mover para baixo
e esquerda
   if (p_posicaox_objeto >= _posicaox && p_
 posicaoy_objeto > _posicaoy)
   {
Mover(3, p_posicaox_maxima, p_posicaoy_maxima);
Mover(2, p_posicaox_maxima, p_posicaoy_maxima);
   }

// Objeto abaixo e à direita - mover baixo e
direita
if (p_posicaox_objeto < _posicaox && p_posicaoy_
objeto > _posicaoy)
   {
Mover(3, p_posicaox_maxima, p_posicaoy_maxima);
Mover(4, p_posicaox_maxima, p_posicaoy_maxima);
       }
   }
}
```

Repare que o método **Aproximar** é bastante semelhante ao método **Escapar**; a espaçonave, nesse caso, se movimenta na mesma direção que o objeto, indo de encontro a ele quando este se aproxima.

Passo 2: Acrescente ao método de atualização do jogo (**Update**) uma instrução para chamar o método **Aproximar** da nave inimiga. O melhor local para fazer isso é logo após a movimentação da espaçonave do jogador. Veja como ficou:

```
if (teclado.IsKeyUp(Keys.Space) && apertou_tiro)
{
   apertou_tiro = false;
}
```

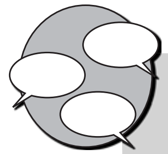
```
// Aproximando o inimigo do jogador
NaveInimiga.Aproximar(NaveJogador.PosicaoX(),
              NaveJogador.PosicaoY(),
              this.Window.ClientBounds.Width,
              this.Window.ClientBounds.Height);

// Atirando contra o jogador
if (NaveInimiga.Atacar(NaveJogador.PosicaoX(),NaveJogador.
PosicaoY()))
{
      NaveInimiga.Atirar(3,graphics.GraphicsDevice,
"../../../Content/Imagens/tiro.png", TirosDisparados);
}
```

Atividade prática 3

Seguindo os dois passos descritos, abra seu projeto de jogo na ferramenta Visual C# e programe o método **Aproximar** da classe Espaçonave para fazer a espaçonave inimiga agir na ofensiva, aproximando-se da nave do jogador quando esta chega perto dela.

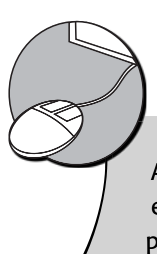
Atividade prática 4

Existe uma pequena falha no código do jogo. Repare que a espaçonave inimiga tenta escapar dos seus próprios tiros. Ajuste o código do jogo de forma que a nave inimiga tente escapar apenas dos tiros disparados pela espaçonave do jogador.
Dica: Será necessário criar na classe Tiro um atributo para indicar quem disparou o tiro.



INFORMAÇÃO SOBRE FÓRUM

Você teve alguma dificuldade para programar a inteligência artificial da espaçonave inimiga dentro do jogo? Entre no fórum da semana e compartilhe suas dúvidas e experiências com os amigos.



Atividade *online*

Agora que você já está com o seu código de projeto do jogo ajustado para tornar a espaçonave inimiga mais inteligente, vá à sala de aula virtual e resolva as atividades propostas pelo tutor.

RESUMO

- **Inteligência artificial** é a programação de ações automáticas a serem tomadas pelos objetos do jogo de acordo com os acontecimentos ocorridos durante o jogo.
- Para tornar a espaçonave inimiga mais inteligente, podemos programar as seguintes ações automáticas com base nas ações que o jogador está tomando durante o jogo:
  - o esquivar-se dos tiros disparados pelo jogador;
  - o atirar na espaçonave do jogador;
  - o aproximar-se da espaçonave do jogador.
- O método mais importante a ser criado na classe Espaçonave é o método **Proximo**. Ele serve para verificar se um objeto está próximo da espaçonave. Esse método é muito importante e será a base para todos os outros testes que iremos fazer. Esse método é bastante semelhante ao método **ColisaoSimples** do jogo.

- Também precisaremos criar os métodos **Escapar**, **Atacar** e **Aproximar**, para que a espaçonave tome as ações já descritas.
  - No método **Escapar**, a espaçonave se movimenta na direção contrária à do objeto em questão;
  - No método **Atacar**, a espaçonave atira contra o objeto em questão, tentando acertá-lo;
  - No método **Aproximar**, a espaçonave se move na mesma direção que o objeto em questão.

SECRETARIA DE
CIÊNCIA E TECNOLOGIA