Microsoft

developer

// Step by step

2D Games with Windows and Windows Phone

FREE SAMPLER

Intermediate



Jebediah Pavleas, Jack Chang,
Kelvin Sung, Robert Zhu

// Step by step

Your hands-on guide to Microsoft XNA Game Studio fundamentals

Expand your expertise—and teach yourself the fundamentals of Microsoft XNA Game Studio for developing games for Windows and Windows Phone. If you have previous programming experience with C# but are new to 2D game programming, each tutorial in this book delivers the step-by-step guidance and coding exercises you need to master core topics and techniques.

Discover how to:

- Explore the conceptual framework behind XNA Game Studio
- Create games for Windows desktops, tablets, and Windows Phone
- Manipulate items and game state via 2D game engine architecture
- Control speed, direction, and pixel-perfect collisions
- Master behaviors such as turning, home-in, chasing, and following
- Bring your characters to life with SpriteSheets, zoom, and audio
- Explode, burn, and blend game elements using particle system effects
- Use the camera and phone sensors in Windows Phone games

About the Authors

Jebediah Pavleas is completing a Master of Science (MS) degree in computer science and software engineering, with interests in game programming and accessibility.

Jack Chang is an experienced game programmer currently completing an MS degree in computer science and software engineering.

Kelvin Sung, PhD, is a professor with the computing and software systems division at a major university.

Robert Zhu, a Principal Development Lead at Microsoft for Windows Phone, is an expert on OS leading edge development, research, and debugging.

Practice Files + Code

Available at:
http://aka.ms/2DGamesWindows/files

Companion eBook

See the instruction page at the back of the book

microsoft.com/mspress

ISBN: 978-0-7356-7718-0

9 780735 677180 90000

U.S.A. $34.99
Canada $36.99
[*Recommended*]

Game Programming/Windows/Windows Phone

Microsoft

Want to read more?

Microsoft Press books are now available through O'Reilly Media. You can buy this book in print and or ebook format, along with the complete Microsoft Press product line.

Buy 2 books, get the 3rd FREE!

Use discount code: OPC10

All orders over $29.95 qualify for **free shipping** within the US.

It's also available at your favorite book retailer, including the iBookstore, the Android Marketplace, and Amazon.com



O'REILLY®

Spreading the knowledge of innovators

oreilly.com

Published with the authorization of Microsoft Corporation by:
O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, California 95472

Copyright © 2013 by Jebediah Pavleas, Jack Chang, Kelvin Sung, Robert Zhu
All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-0-7356-7718-0

1 2 3 4 5 6 7 8 9 LSI 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at *mspinput@microsoft.com*. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Microsoft and the trademarks listed at *http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx* are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, O'Reilly Media, Inc., Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions and Developmental Editor: Russell Jones

Production Editor: Kristen Brown

Editorial Production: Zyg Group, LLC

Technical Reviewer: John Mueller

Copyeditor: Damon Larson

Indexer: BIM Indexing Services

Cover Design: Twist Creative • Seattle

Cover Composition: Ellie Volckhausen

Illustrator: Rebecca Demarest

Contents

Foreword . . . . . *xi*
Introduction . . . . . *xiii*

Chapter 1 Getting started with XNA 1

XNA Game Studio and Silverlight 1
Choosing between XNA and Silverlight 2
Downloading and installing development tools 2
Windows 8 and XNA 3
Visual Studio 2012 and XNA 3
Coverage of game development topics 5
Technology references 8

Chapter 2 Game development tools and XNA 9

Introduction 9
The Visual Studio development environment 10
The Visual Studio layout and Solution Explorer 12
The relationship between the file system and Solution Explorer . . 15
Understanding the XNA Framework 16
XNA.Framework.Game functions 18
Adding, drawing, and controlling content 20
The Draw and Control project 20
Summary 25
Quick reference 26

What do you think of this book? We want to hear from you!
Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

Chapter 3 Graphics and effects, part 1 27

Introduction 27

The game window 27

The Game Window Size project 28

Textured primitives 30

The Textured Primitive project 30

Observations 38

Coordinate system and camera 39

The User-Defined Coordinate System project 39

Font output 48

The Font Output project 49

A simple game object 53

The Simple Game Object project 53

Simple game state 61

The Simple Game State project 61

Summary 69

Quick reference 70

Chapter 4 Math and game object behaviors 71

Rotating textures 71

The Rotate Textured Primitive project 71

Vectors 77

Vector review 77

The Show Vector project 79

Front direction 87

The Front Direction project 87

Game objects 91

The Game Object project 91

Chasers 98

The Chaser Object project 98

Summary 105

Quick reference 106

Chapter 5 Pixel-accurate collision and simple physics 107

Pixel-accurate collision 107
The Pixel-Accurate Collision project 108
General pixel collision 115
The General Pixel Collision project 115
Simple physics 120
The Simple Physics project 121
Summary 128
Quick reference 128

Chapter 6 Game object states 129

Review of finite state machines 129
Creating a spinning arrow 130
The Spinning Arrow project 130
Adding many spinning arrows 137
The Many Spinning Arrows project 137
Creating a patrol enemy 142
The Patrol Enemy project 142
Implementing smooth turning 149
The Smooth Turning Patrol project 150
Patrol that chases 153
The Patrol That Chases project 153
Creating many enemies 158
The Many Enemies project 159
Summary 161
Quick reference 162

Chapter 7 Graphics and effects, part 2 163

Sprite animation 163
The Sprite Animation project 163
Sprite collision 172
The Sprite Collision project 172

Moving and zooming the camera 179
The Camera Zoom Move project 179
Adding audio 185
The Audio project 185
Summary 190
Quick reference 190

Chapter 8 Particle systems 193

Particle systems 193
The Particle System project 193
Particle emitters 201
The Particle Emitter project 201
Summary 206
Quick reference 206

Chapter 9 Game development for the Windows Phone platform 207

Graphics basics on Windows Phone 208
2D graphics basics on the Windows Phone platform 208
Drawing graphics on Windows Phone 209
Windows Phone 8: When to choose XNA to develop 2D games . 209
Creating a 2D XNA game on Windows Phone 210
Inside the 2D XNA game engine on Windows Phone 212
Adding animation to Windows Phone games 219
Windows Phone peripherals 223
Using the camera 224
Using sensors 230
Accessing networks 236
Summary 237
Quick reference 238

Index *239*

About the authors *251*

Introduction

Welcome to game development for Microsoft Windows and Windows Phone with Microsoft XNA Game Studio. Because you have picked up this book, you are likely interested in creating your own games for the Windows and Windows Phone platforms with the C# programming language. This book teaches you how to develop 2D games with XNA by giving you background and conceptual information so you can play, examine, and develop 2D games.

This book identifies and presents relevant concepts from software engineering, computer graphics, mathematics, physics, and game development—all in the context of building 2D games. The projects you'll develop in this book are based on the popular XNA Framework. The presentations are tightly integrated with the analysis and development of source code; you'll spend much of the book building gamelike concept projects that demonstrate game principles and components. By building on concepts introduced early on, the book leads you on a journey through which you will master the basic concepts behind game development while simultaneously gaining hands-on experience developing simple but working 2D games.

By the end of the book, you will be familiar with the implementation details of 2D games, and you should feel competent in implementing commonly encountered 2D game behaviors using the XNA Framework.

This book also touches on using the sensors available on Windows Phone and RT devices and will help you take advantage of those sensors in your XNA games.

Who should read this book

This book is targeted toward programmers who are familiar with basic object-oriented programming concepts and have a basic to intermediate knowledge of an object-oriented programming language like C# or Java. For example, if you are a student who has taken a few introductory programming courses, an experienced developer who is new to games and graphics programming, or a self-taught programming enthusiast, you will be able to follow the concepts and code presented in this book with little trouble. If you're new to programming in general, it is suggested that you first become comfortable with the C# programming language before tackling the content provided in this book.

Assumptions

You should be experienced with programming in an object-oriented programming language, such as C# or Java. The examples in this book were created with the assumption that readers understand data encapsulation and inheritance. In addition, you should also be familiar with basic data structures such as lists and dictionaries, and be comfortable working with the fundamentals of algebra and geometry, particularly linear equations and coordinate systems.

Who should not read this book

This book is not designed to teach readers how to program, nor does it attempt to explain intricate details of C# or XNA. If you have no prior experience developing software with an object-oriented programming language, you will probably find the examples in this book difficult to follow.

On the other hand, if you have an extensive background in game development for other platforms and with other programming languages, the content here will be too basic; this is a book intended for developers without 2D game development experience; however, you might still pick up a few useful tips about 2D game development for the platforms covered in this book.

Organization of this book

This book divides the process of building 2D games into essential topic areas: tools, graphics, special effects, math and physics, and logic and behavior. These topics are organized into chapters. Each topic area (chapter) is then subdivided into essential concepts; for example, concepts related to computer graphics include coordinate spaces and camera abstraction. The book introduces each concept via a gamelike example organized as a section in a chapter. Each has an associated step-by-step project workflow. In this way, each section in the book corresponds to a single project or concept.

The first section begins with a simple project that you will build from scratch. Throughout the text, each subsequent section builds upon the sections that precede it. While this makes it a bit difficult to skip around in the book, it will give you practical experience and a solid understanding of how the different concepts relate to one another. In addition, rather than always working with new and minimalistic projects, you gain experience with building larger and more interesting projects.

The projects themselves start with simple concepts, such as creating objects and moving them across the screen, but quickly move to more complex concepts, such as implementing pixel-accurate collision detection and working with user-defined coordinate systems. In this way, while the concepts are presented in simple 2D gamelike examples, by the end of the book, your code base for the projects will include all the essential concepts covered.

That final code base, which you will have developed incrementally over the course of the entire book, serves as a great platform on which you can begin building your own 2D games.

Finding your best starting point in this book

As a reader, there are several ways for you to follow along in this book. The first and most obvious is to enter the code into your project as you follow through each step in the book. From a learning perspective, this is probably the most effective way to absorb the information presented; however, we understand that it may not be the most realistic, due to the amount of code or debugging that approach may require. To help ameliorate this, each section and project in this book has two corresponding source code folders: the starter project in a 1.Starting folder, and the completed project in a 2.Completed folder. You can see an example of this structure in Figure 1. The starter projects allow you to follow along in the corresponding section by entering the code as you encounter it in the book, while the completed project lets you run and see the project in its completed state.

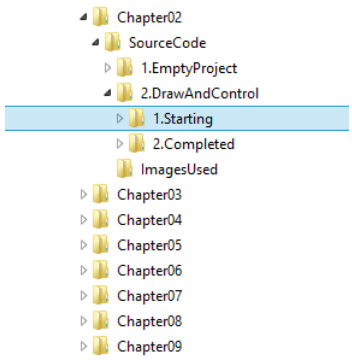


FIGURE 1 The folder structure for the book projects

We recommend that you refer to the completed project when you begin a new section. Doing so lets you preview the current section's project, giving you a clear idea of the end goal, and letting you see what the project is trying to achieve. You may also find the completed project code useful when you have problems while building the code yourself, because you can compare your code with the completed project's code during difficult debugging situations.

Finally, after completing a project, we recommend that you compare the behavior of your implementation with the completed-project implementation provided. By doing so, you can observe whether your code is behaving as expected.

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow:

- Each example is built on top of prior ones; however, complete source code is provided for each exercise, so you can use that to skip sections you don't need.
- A screen shot and an outlined overview of the steps involved precede each step-by-step procedure, so that you will know what to expect.
- When necessary, we provide relevant background information before beginning the analysis of how to implement the concept. For example, the book discusses differences between pixel and user-defined spaces before analyzing how to implement a user-defined coordinate system.
- Source code analysis is divided into distinct steps, where each step contributes to the eventual implementation of the concept.
- Items that you should type (except source code) appear in bold text.
- Programming-related items, such as class names, variable names, namespaces, and so on, appear in italics.
- When you need to press two keys at once, such as holding the Control key while pressing the A key, this book shows the two keys separated by a plus (+) sign—for example, Ctrl+A.

System requirements

You will need the following hardware and software in order to follow the examples in this book:

- Windows 8
- Microsoft Visual Studio 2010 and 2012, any edition
- XNA 4.0 and the Windows Phone 8 SDK
- A computer that has a 1.6 GHz or faster processor (2 GHz is recommended)
- More than 1 GB of RAM
- 1 GB of available hard disk space
- A Microsoft DirectX 9–capable video card that can run at a resolution of 1,024×768 or higher
- An Internet connection, for downloading software or chapter examples

Most of the projects in this book were built to use a Microsoft Xbox 360 controller for Windows. You should be able to work with any wired Xbox 360 controller by plugging it into a USB port on your computer; however, wireless Xbox 360 controllers require an Xbox 360 wireless receiver for Windows to function on the PC. If you instead want to use a keyboard and mouse with your Windows-based projects, you will need to make some code modifications. The types of changes you'll need to make are addressed near the end of the Draw and Control project in Chapter 2, "Game development tools and XNA." Chapter 9, "Game development for the Windows Phone platform," does not rely on an Xbox controller; instead, it uses the input devices provided by the Windows Phone hardware.

You can find details on how to download, install, and configure Visual Studio, XNA, and the Windows Phone SDK in Chapter 1, "Getting started with XNA."

Code samples

With the exception of the first chapter, all the chapters in this book include examples that let you interactively experiment with and learn the new materials. You can download all the code for all the projects, including the associated assets (images, audio clips, or fonts), in both their pre-example and completed states from the following page:

http://aka.ms/2DGamesWindows/files

To install the code samples, unzip the 2DGamesWithXNA_sample_code.zip file. You should see the folder structure shown previously in Figure 1. As described previously, the starter project is in the 1.Starting folder, and the corresponding completed project is in the 2.Completed folder. With Visual Studio properly installed, you can double-click the corresponding solution (.sln) file to begin working with any of the provided projects.

Acknowledgments

The authors of the book would like to thank Darren Laybourn for taking the time to write the foreword.

The genesis of this book initially occurred as part of the XNA Based Game-Themed Programming Assignments for CS1/2 project, funded under the Computer Gaming Curriculum in Computer Science initiative from Microsoft External Research and by Microsoft Research Connections. We would like to thank John Nordlinger for his recognition of our vision, and Donald Brinkman and Lee Dirks for their continual support, including invaluable discussions and for allowing us incredible access. Throughout the project, Kent Foster has been our best advocate, connecting us with many end users who provided much-needed refinements to the examples. The final refinement of the materials and the book development efforts are part of the Game-Themed CS1/2: Empowering the Faculty project, funded by the Transforming Undergraduate Education in Science Technology Engineering and Mathematics (TUES) Program (National Science Foundation [NSF] award number DUE-1140410).

Students and the games they build from the course CSS385: Introduction to Game Development (see *http://courses.washington.edu/css385)* at the University of Washington, Bothell have provided us with the ideal deployment vehicle and are a source of continuous inspiration. They have tested, retested, contributed to, and assisted in the formation and organization of the contents. The first two authors of this book are recent alumni of CSS385.

The University of Washington, Bothell logo, which is used throughout this book, is a trademark owned by the University of Washington, Bothell. The permission to use the logo is a courtesy of and does not represent endorsement from the institution. The Mind_Meld.mp3 audio clip used in the audio project from Chapter 7, "Graphics and effects, part 2," was composed by Shane Krolikowski (*http://skrolikowski.com/*) using Apple's GarageBand samples.

We had the privilege of getting input from Laura Butler for materials related to Windows Phone. She is a walking animated encyclopedia about Windows and Windows Phone graphics, imaging, input, messaging, and camera features. We are truly delighted to be able to share some of her knowledge with readers, and we feel fortunate indeed, given her busy schedule, that Laura was able to review the content related to Windows Phone.

Additionally, for our Windows Phone content, we want to thank Tom Adam, Ying Chin, Saty Bahadur, Kevin Lee, Charlie Chase, Shashank Gupta, Tony Zhao, Sunny Cheng, Dana Huang, and Dan Conley for their valuable comments.

We also want to thank Pei Zheng for introducing us to Russell Jones, our editor from Microsoft Press. A heartfelt thank you to Russell for his patience and expertise in guiding us through the book-writing process, and many thanks to John Mueller, our technical reviewer. Finally, we would like to thank Kara Ebrahim, Kristen Brown, Devon Musgrave, and Damon Larson for their insightful technical feedback and writing guidance.

Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site at oreilly.com:

http://aka.ms/2DGamesWindows/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

CHAPTER 1

Getting started with XNA

It is no secret that developing games can be a challenging and lengthy process. This is partly due to the general programming knowledge and experience required to begin game development, and partly due to the steep learning curve associated with most graphics APIs, such as Microsoft DirectX and OpenGL. Microsoft XNA Game Studio addresses these issues by creating a developer-friendly graphics framework with a much smaller learning curve. This framework lets developers quickly learn the information needed to begin creating 2D games for Microsoft Windows and Windows Phone. Another developer-friendly framework for Windows and Windows Phone that also provides a shorter learning curve for application development is Microsoft Silverlight. While this book concentrates on XNA development, it's worth taking a quick look at the capabilities of both XNA and Silverlight.



Note If you wish to learn more about or brush up on the technologies discussed in this chapter, such as C#, XNA, DirectX, or OpenGL, see the "Technology references" section at the end of this chapter.

XNA Game Studio and Silverlight

Silverlight is a development and delivery environment built on top of the .NET Compact Framework, and intended for creating applications that run on the Web and on mobile devices, as well as the desktop. Silverlight for the desktop is somewhat limited compared to native desktop applications because of its initial design as a web-based API. However, if you already have experience developing apps in Silverlight for web or desktop use, you'll find it easy to develop applications for both Windows Phone 7 and 8, because the Silverlight framework on Windows Phone is a subset of the .NET Framework development platform with added phone-specific APIs. Additionally, the Silverlight subset has been integrated into Visual Studio to further simplify developing Silverlight applications.

XNA Game Studio is yet another API based on the .NET Framework for creating applications. You can deploy XNA applications on Windows, Windows Phone, and Microsoft Xbox 360. In contrast to Silverlight, the XNA Framework was specifically designed for developing games. It provides a *game loop*, which you can control to update game values and draw to the window, as well as a content pipeline for importing assets such as images and audio files. It also takes advantage of the graphics processing unit (GPU) to render graphics on the screen rapidly and smoothly.

Choosing between XNA and Silverlight

The best choice between XNA and Silverlight depends on the type of application you are creating. For mobile-based and web-based software applications, Silverlight is an easy choice. Its event-driven framework and UI creation provide developers with the tools they need to create apps quickly and easily.

On the other hand, XNA is usually the better choice for creating 2D or 3D game applications. This is because XNA provides the API functionality that most games require, including (but not limited to) access to a game loop and a content pipeline, and support for fast graphical manipulations (such as drawing only selected portions of an image and implementing sprite sheet animation).

This makes the choice of frameworks for this book easy; however, this does not mean that you can't create games in Silverlight—or for that matter, that you can't create software applications other than games in XNA. Instead, the point is that using a framework that was not designed to support what you're developing is likely to cause unneeded headaches.

It's worth noting that when you use the Silverlight framework, you can still call many XNA Framework APIs—except those in the *Microsoft.Xna.Framework.Game* class and the *Microsoft.Xna.Framework.Graphics* namespace. Similarly, when developing in the XNA Framework, you can also use many classes in the Silverlight framework—except many (but not all) of the classes in the *System.Windows* namespace.



Note For more information on the classes that the *Microsoft.Xna.Framework* namespace provides, see the links in the "Technology references" section at the end of this chapter.

Downloading and installing development tools

To use XNA and follow along with this book, you will need the Windows 7 or Windows 8 operating system and three additional pieces of software:

- The Microsoft Visual Studio IDE (integrated development environment)
- The XNA Game Studio 4.0 SDK (software development kit)
- The Windows Phone SDK

The first piece of software you need is the IDE. To develop in XNA, you will need to use a version of Visual Studio. The examples in this book use Visual Studio 2012 Express, which is free to download. However, none of the Visual Studio 2012 versions support XNA out of the box; you need to take some additional steps before installing the software, as detailed following. You can find Visual Studio 2012 Express (and other versions) at the following link:

http://www.microsoft.com/visualstudio/downloads

You will also need the XNA Game Studio 4.0 Refresh, which you can download here:

http://www.microsoft.com/download/details.aspx?id=27599

Lastly, to follow along with the examples in Chapter 9, "Game development for the Windows Phone platform," and develop for Windows Phone 8 devices with the Windows Phone SDK, you must use Windows 8 in conjunction with Visual Studio 2012. If you do not have the Windows 8 operating system or a version of Visual Studio 2012, the examples in Chapter 9 of this book will not work. You can download the Windows Phone SDK here:

http://www.microsoft.com/download/details.aspx?id=35471

Windows 8 and XNA

If you're using Windows 8 and you get an error when installing the XNA Game Studio 4.0 Refresh, follow these steps:

1. Download and install the Games for Windows Marketplace client here:

   http://www.xbox.com/LIVE/PC/DownloadClient

2. Download and install the Microsoft XNA Game Studio 4.0 Refresh again here:

   http://www.microsoft.com/download/details.aspx?id=27599

Visual Studio 2012 and XNA

To use Visual Studio 2012 with XNA, you also need to have Visual Studio 2010 (the Express version will work fine) and XNA 4.0 installed on your machine. You will need to manually copy the IDE template by following the instructions provided next. Note that there are two separate sets of instructions: one for Visual Studio 2012 Express and one for Visual Studio 2012 Professional or Visual Studio 2012 Ultimate. Follow the procedure that corresponds to the version of Visual Studio you are using.



Note The following Visual Studio setup procedures assume that you're using a 64-bit version of Windows 7 or 8 and a 32-bit version of Visual Studio. However, if you are using a 32-bit version of Windows or a 64-bit version of Visual Studio, then you should replace all instances of "\Program Files (x86)\" with "\Program Files\".

Setup for Visual Studio 2012 Express

1. Open a command prompt (cmd.*exe*) as an administrator by selecting Start | All Programs | Accessories, and then right-clicking the Command Prompt icon and choosing Run As Administrator. The title of the resulting command window should read Administrator: Command Prompt.

2. Type the following command and run it by pressing Enter:

```
xcopy /e "C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\Extensions\
Microsoft\XNA Game Studio 4.0" "C:\Program Files (x86)\Microsoft Visual Studio 11.0\
Common7\IDE\WDExpressExtensions\Microsoft\XNA Game Studio 4.0"
```



Note Visual Studio's default installation path is on the C drive. If you installed Visual Studio on a different drive, you will need to modify the path to reflect your system setup.

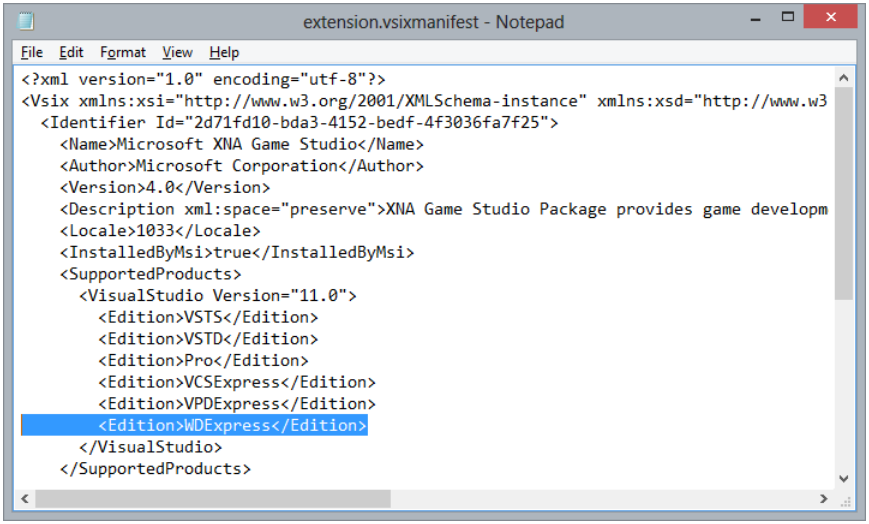
3. When asked, "Does [destination] specify a file name or directory name on the target (F = file, D = directory)?" answer with "D."

4. Open Windows Explorer and navigate to C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\Extensions\Microsoft\XNA Game Studio 4.0.

   a. Open extension.vsixmanifest with a text editor such as Notepad.

   b. Locate the line <*VisualStudio Version="10.0"*> and change it to <*VisualStudio Version="11.0"*>.

   c. Add the line <*Edition*>*WDExpress*</*Edition*> where it is shown in the image that follows.



5. Navigate to your %LocalAppData%\Microsoft\WDExpress\11.0\Extensions folder, and delete any files that fit the patterns extensionSdks.*.cache and extensions.*.cache.

6. Open a command prompt (cmd.exe) as an administrator, or use the one you opened in step 1 if it's still open.

7. Input the following command and press Enter to run it:

```
"C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\WDExpress.exe" /setup
```

Setup for Visual Studio 2012 Professional/Ultimate

1. Open a command prompt (cmd.exe) as an administrator by selecting Start | All Programs | Accessories, right-clicking the Command Prompt icon, and then choosing Run As Administrator. The title of the resulting command window should read Administrator: Command Prompt.

2. Input the following command and run it by pressing Enter:

```
xcopy /e "C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\Extensions\
Microsoft\XNA Game Studio 4.0" "C:\Program Files (x86)\Microsoft Visual Studio 11.0\
Common7\IDE\Extensions\Microsoft\XNA Game Studio 4.0"
```



Note Visual Studio's default installation path is on the C drive. If you installed Visual Studio on a different drive, you will need to modify the path to reflect your system setup.

3. When asked, "Does [destination] specify a file name or directory name on the target (F = file, D = directory)?" answer with "D."

4. Open Windows Explorer and navigate to the folder C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\Extensions\Microsoft\XNA Game Studio 4.0.

   a. Open the extension.vsixmanifest file with a text editor such as Notepad.

   b. Locate the line <*VisualStudio Version="10.0"*> and change it to <*VisualStudio Version="11.0"*>.

5. In Windows Explorer, navigate to your %LocalAppData%\Microsoft\VisualStudio\11.0\Extensions folder, and delete any files that match the patterns extensionSdks.*.cache and extensions.*.cache.

6. Open a command prompt (cmd.exe) as an administrator, or use the one you opened in step 1 if it's still open.

7. Input the following command and run it:

```
"C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\devenv.exe" /setup
```

Coverage of game development topics

Because this book is targeted toward individuals with some experience in software development, it jumps headlong into walking you through the technical details you need to know to create your own games from scratch. The overall goal is to guide you through this process as quickly and painlessly as possible while providing a solid foundation in the game development concepts used in the industry today.

Also note that *game development* is a catchall term that actually includes many different disciplines. While some of these disciplines lie within the computer science and computer programming fields, there are several other areas of game development that are actually outside of the computer science and programming disciplines, including areas such as game design and asset creation (art). These areas are as crucial to the success of a game as the technical process. With that in mind, we'll detail the areas of development that are and are not covered in this book. The following list gives an overview of those topics that are covered:

- Development tools (the Microsoft Visual Studio IDE and the XNA Framework):
  - Installation guide
  - Conceptual framework behind XNA
- The basic architecture of a 2D game engine:
  - Draw and update cycles
  - Assets and resource-loading models
  - Game object abstraction and encapsulation
  - Game state implementation
- Math and physics:
  - Drawing coordinate systems
  - 2D camera abstraction
  - Working with randomness
  - Points, vectors, dot product, and cross product, and their support for implementing speed, velocity, direction, collision responses
- Logic and behaviors:
  - Finite state machines
  - Game complexity and linear collections
  - Pixel-accurate collisions
  - Semiautonomous behaviors (controlled gradual turning, homing in, chasing, and following)
- Graphics and effects:
  - 2D camera manipulation
  - Sprite sheets and sprite animations
  - Particle systems
  - Fonts and audio effects

- Windows Phone:
  - XNA projects for Windows Phone devices
  - Working with sensors on Windows Phone (microphone, camera, and accelerometer)

The following list describes those areas of development that *are not* covered in the book:

- **The development process** This encompasses the entire life cycle of game creation—everything from the initial idea to the finishing touches. There are many life-cycle processes in use today; one commonly used for many software projects is agile development.
- **Game design** This is the phase of brainstorming and creating the components and parts that comprise the game. In general, it is accepted that the game design should be documented before the process of building the game begins; however, there are many different approaches to creating the game design document itself. Some people prefer to flesh out all the mechanics the game will have in a single large document, while others prefer to create a concise concept document of a few pages, and then begin prototyping before completely fleshing out the mechanics.
- **Game architecture planning** This details how the game will be structured, including what objects the game will include, and how they will interact with one another. This phase relies heavily on knowledge of object-oriented programming.
- **Asset creation** This is the creation and preparation of assets (art, video, and audio) for the game. Asset creation includes everything from the process of creating sprite sheets to properly sizing images.

If you're interested in some of these other areas of the game development process, or wish to dive more deeply into the technical aspects of creating 2D or 3D games, a plethora of resources exist. The authors have found the following books on game design and the development process to be very helpful and informative:

- *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*, by Tracy Fullerton (CRC Press, 2008)
- *Fundamentals of Game Design (2nd Edition),* by Ernest Adams (New Riders, 2009)
- *The Art of Game Design*, by Jesse Schell (CRC Press, 2008)

While the step by step tutorials from this book cover many of the foundational concepts in typical 2D game engines, the above areas can help you with the design and implementation of fun games. Together these form a concrete foundation for building simple 2D games. In order to begin to understand and appreciate the intimate details of popular games like the Halo series, it helps to have the technical knowledge from standard computer science and computer engineering undergraduate curricula, including computer graphics, classical mechanics, artificial intelligence, networking, databases, human computer interaction, software engineering, and so on. In addition, many of the advanced autonomous behaviors in games utilize concepts from linear algebra.

Technology references

The following list offers several links for obtaining additional information on general topics both related and unrelated to this book.

- **C#** *http://msdn.microsoft.com/library/vstudio/kx37x362.aspx*
- **DirectX** *http://msdn.microsoft.com/library/ms810424.aspx*
- **OpenGL** *http://www.opengl.org/*
- **XNA Framework reference** *http://msdn.microsoft.com/library/bb203940.aspx*
- **Silverlight** *http://msdn.microsoft.com/silverlight/bb187358.aspx*
- **Agile development** *http://www.agilealliance.org/the-alliance/the-agile-manifesto*

CHAPTER 2

Game development tools and XNA

After completing this chapter, you will be able to

- Use the most important features of the Microsoft Visual Studio integrated development environment (IDE)
- Use the Microsoft XNA Game Studio project template
- Describe the functions of all the relevant files in an XNA project
- Work with the XNA Game Studio project and application structure
- Use the functions and operations of the Draw/Update loop

Introduction

Before diving headlong into the creation of games for Microsoft Windows and Windows Phone, it is a good idea to first become familiar with the development environment you will be using. The primary tools you will work with throughout this book are Visual Studio Express, XNA Game Studio, and the Windows Phone Developer Tools. As you may already know, Visual Studio is the IDE you will be using for development. XNA Game Studio is a framework created by Microsoft that provides developers with software libraries and reusable tools designed specifically for game development. Similarly, the Windows Phone Developer Tools comprise a tool set designed to assist in the development of Windows Phone applications.

Most of the fundamental concepts behind programming games with XNA are independent from those of Windows Phone development. For this reason, in this book we will first focus on learning the details of XNA before discussing the Windows Phone Developer Tools. For this chapter, we will focus on Visual Studio and XNA.

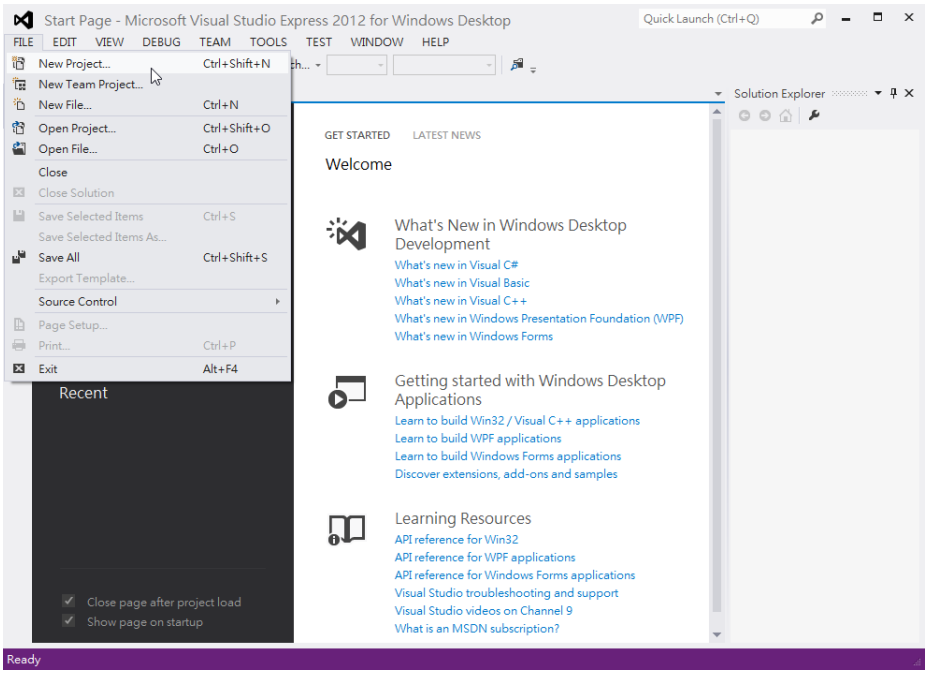
The Visual Studio development environment

To follow along, you will need to download and install both Visual Studio and XNA Game Studio. If you have yet to install either of these packages, you can refer to the download and installation instructions in the "Downloads and installations" section of Chapter 1, before proceeding.

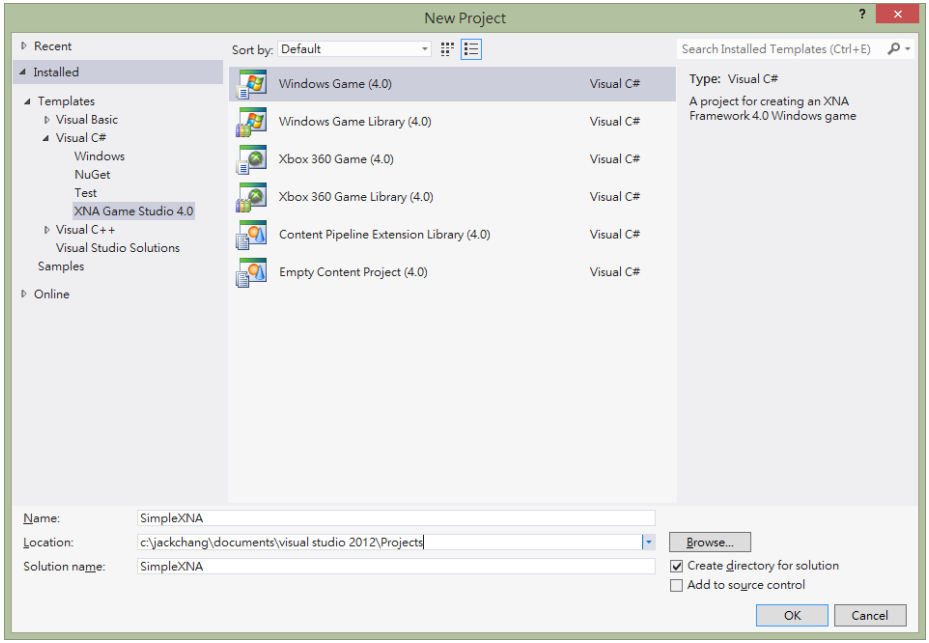
After you have installed the required tools, you can start creating your first project. Luckily, Visual Studio Express with XNA Game Studio lets you create new projects with minimal effort.

Creating an XNA project in Visual Studio

1. Open Visual Studio. Press Ctrl+Shift+N on your keyboard or select File | New Project, as shown in the following image. A New Project window will appear.



2. Under the Installed Templates section, expand the Visual C# section, and then select XNA Game Studio 4.0. A list of templates for both Windows and Microsoft Xbox 360 will appear. You can see an example of this in the following image:



3. Select Windows Game (4.0) from the Installed Templates. A short description of the template will appear on the right.

4. Name the project SimpleXNA.



Note Optionally, you can also specify the solution name of the project. (By default, the solution name will be the same as the name of the project.)

5. If you like, you may specify a location for the project on your file system (the default location is in the Projects folder under C:\YourUserName\Documents\Visual Studio 2012\Projects).

6. Click the OK button.

Visual Studio generates an XNA project named SimpleXNA for you. The project contains several default files that in turn contain a minimum skeleton of code required for the project to compile and run.

Go ahead and start the program now by selecting Debug | Start Debugging or by pressing the F5 key on your keyboard. A new window will appear showing an empty game world in cornflower blue. Figure 2-1 shows an example of what the default project (which has no real content yet) looks like when you run it.

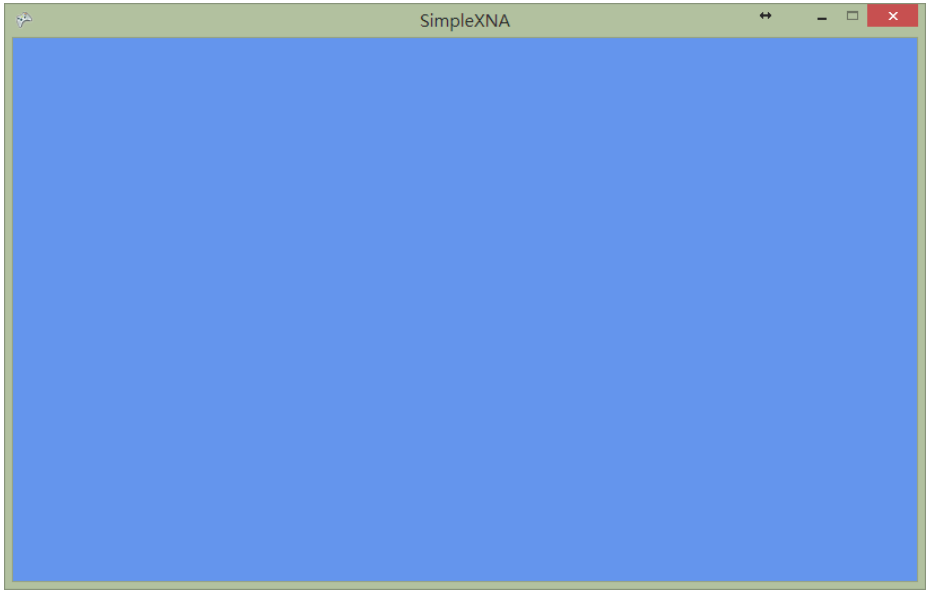


FIGURE 2-1 Running the default game project

To stop the program, either close the game world window or select Debug | Stop Debugging in Visual Studio.

You have successfully run your first XNA project. Now that you have a working project, you'll use that to help you get oriented with the IDE.

The Visual Studio layout and Solution Explorer

Upon first examining the Visual Studio development environment shown in Figure 2-2, you'll notice it is divided into four main sections:

- **Primary window** Displays the source code or the information of the currently selected file
- **Error List window** Displays the syntax errors in the code
- **Solution Explorer window** Displays the projects and files for the current solution
- **Properties window** Displays the properties of the currently selected object

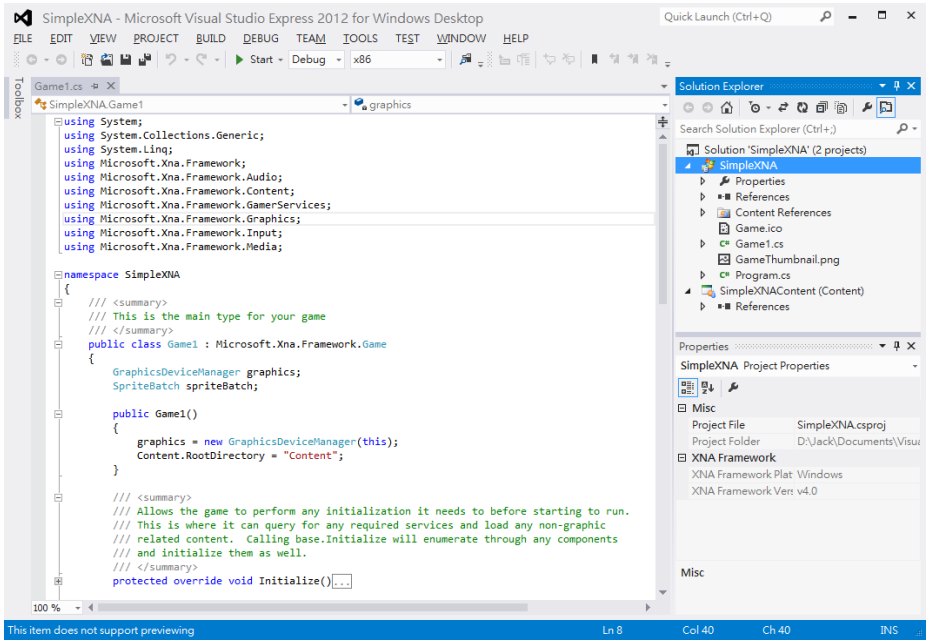


FIGURE 2-2 The Primary, Error List, Solution Explorer, and Properties windows of Visual Studio



Note If you do not see the Error List, Solution Explorer, or Properties window in your IDE, click the View menu and then select the name of the missing window to display it in the IDE.

Because Solution Explorer is the least straightforward window in the bunch, it's worth exploring it in more detail. In Figure 2-3, you can see that the SimpleXNA solution contains two projects (even though you only created one in the previous procedure): a SimpleXNA project and a SimpleXNAContent project. These projects represent the game project itself and the game's content (the game assets, such as art and sound). The two projects are tied together by the content references in the SimpleXNA project. You can see the reference to the SimpleXNAContent project in the following image:

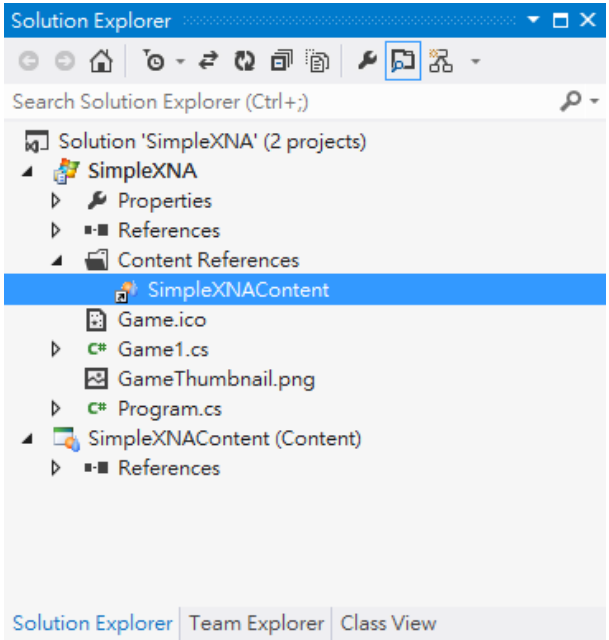


FIGURE 2-3 The Solution Explorer window

Table 2-1 describes the functions of the remaining files shown in Solution Explorer.

TABLE 2-1 Purpose of each file

SimpleXNA project: folder/file	Purpose
Game.ico	Icon image in the top-left corner of the application window
Properties/AssemblyInfo.cs	Configuration and information of the game—for example, the window title and the Globally Unique Identifier (GUID), which gives the game a unique name
Program.cs	Container of the *main()* function
Game1.cs	Source code of the game
GameThumbnail.png	Thumbnail image for Xbox 360 and Windows Phone
References	Reference to all required system libraries (including XNA libraries)
Content Reference	Reference to the content project
Content Project	No files, only references to libraries

As you develop the game, you'll modify most of these files to suit the needs of your game, especially the file Game1.cs. Along with the source code files you create, Game1.cs is where the logic of your game resides. However, before tackling the source code in more detail, take a look at the project you created in your file system so you can see its relation to Solution Explorer.

The relationship between the file system and Solution Explorer

Open the SimpleXNA project location on your file system. The folder should look similar to the one shown in Figure 2-4. Let's examine these files a little more closely. Visual Studio generates both the SimpleXNA.sln and SimpleXNA.suo files when you create the project. SimpleXNA.sln is the project solution description, while SimpleXNA.suo records various Visual Studio settings associated with the current solution.

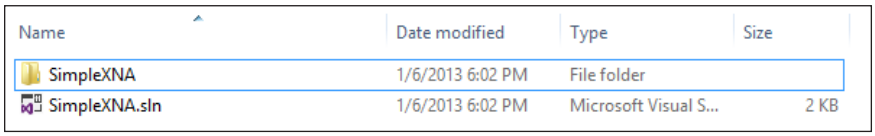


FIGURE 2-4 Contents of the SimpleXNA project folder

By opening the SimpleXNA\SimpleXNA subfolder, you will see that it contains the two respective projects (the folders) you saw in Visual Studio's Solution Explorer. Figure 2-5 reflects this.

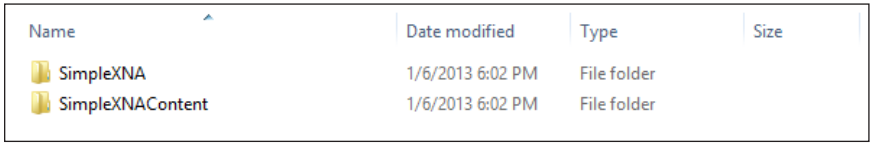


FIGURE 2-5 Contents of the SimpleXNA\SimpleXNA project folder

Similarly, by opening the SimpleXNA\SimpleXNA\SimpleXNA project folder, which is shown in Figure 2-6, you can see many of the same files that were shown by Solution Explorer—along with a few additions. The bin and obj folders are hidden by default in Solution Explorer. They contain the project's executable and compiled files, respectively. The SimpleXNA.csproj file contains the project description and is also hidden by Solution Explorer. The remaining files all appear within Solution Explorer and were described in Table 2-1.



Note To display the hidden files within Solution Explorer, toggle the Show/Hide button in the top-left corner of the Solution Explorer window.

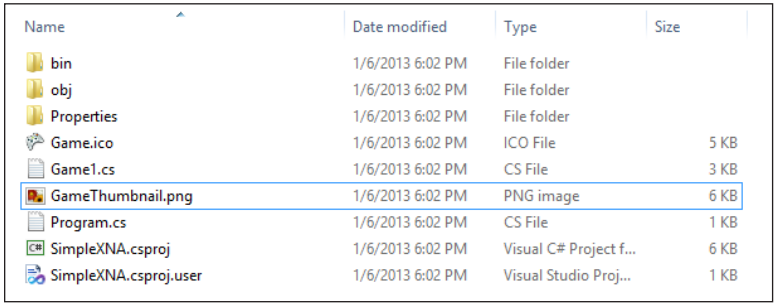


FIGURE 2-6 Contents of the SimpleXNA\SimpleXNA\SimpleXNA project folder

The first file you should know about is AssemblyInfo.cs. You can find that file in the Properties folder in Solution Explorer. Overall, AssemblyInfo.cs is not the most interesting source file. However, there are a few important aspects that we can touch upon, the first being the title of the game window. The following code shows the contents of the file. The first line reads *[assembly: AssemblyTitle("SimpleXNA")]*. This code, which was generated automatically, sets the game window's title to *SimpleXNA* when the project executes. Following this line are many other descriptors for the project, including company and trademark. Also, note the line of code that reads *[assembly: Guid("2654e9a6-aa9c-4675-8bf5-83e4ec038421")]*, which represents the unique ID generated for this project.

```
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("SimpleXNA")]
[assembly: AssemblyProduct("SimpleXNA")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyCompany("Microsoft")]
[assembly: AssemblyCopyright("Copyright © Microsoft 2011")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components. If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type. Only Windows
// assemblies support COM.
[assembly: ComVisible(false)]

// On Windows, the following GUID is for the ID of the typelib if this
// project is exposed to COM. On other platforms, it unique identifies the
// title storage container when deploying this assembly to the device.
[assembly: Guid("4174a8c6-5e9b-4822-b812-be206e0e29da")]
```

The next file of interest is Program.cs. This is the source file that runs your game. Another way to look at it is that this file contains the *main()* function, or entry point for code execution. First, at the top of the following code is the *using System;* statement. For those with experience in other languages, the keyword *using* in this line is similar to the keywords *include* of C++ and *import* of Java. As you look inside *main()*, you will see that the code instantiates a *Game1* object and invokes its *Run()* command.



Note When starting out, it is a good idea to avoid making modifications to this file. Try to keep all the game code in the *Game1* object and other files you create.

```
using System;

namespace XNABook_Example
{
#if WINDOWS || XBOX
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
#endif
}
```

The last project-generated source code file you should look at is Game1.cs. A shortened example of Game1.cs is shown following. First, note that the file inherits from (is a subclass of) the *Microsoft.Xna.Framework.Game* class. This provides the *Game1* class with access to code designed to assist in common game tasks, such as initialization, rendering, loading, and updating. You can see this reflected in the override functions that were generated for you (shown following). These functions are invoked by the *Xna.Framework* library in a fixed sequence to support proper initialization and runtime functionality. Because you will be using these *Xna.Framework* functions often, you'll explore them in more depth in upcoming sections.

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

        public Game1()
        {
            ...
        }

        protected override void Initialize()
        {
            ...
        }

        protected override void LoadContent()
        {
            ...
        }
```

```
        protected override void UnloadContent()
        {
            ...
        }

        protected override void Update(GameTime gameTime)
        {
            ...
        }

        protected override void Draw(GameTime gameTime)
        {
            ...
        }
    }
}
```

XNA.Framework.Game functions

Like most classes, a *Microsoft.Xna.Framework.Game* subclass starts with a constructor. The constructor is responsible for allocating and initializing the internal memory of the game. It also handles the internal initialization of XNA by creating and initializing the graphics hardware. You can see this reflected in the following code, which shows the constructor of the *Game1* class with the line *graphics = new GraphicsDeviceManager(this)*. This line causes XNA to obtain and prepare the graphics device on your machine so you can use it for your game.

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

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        ...
    }
}
```

The *Initialize()* function gets called next. This function initializes the game's graphics requirements, such as the window size. However, by default, the only call within the *initialize* function is its superclass function call, *base.Initialize()*, as shown in the following example. As you will see in many of the examples in the rest of this book, you will typically add custom nondefault initialization code in this function.

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

    base.Initialize();
}
```

After the initialize function, XNA calls the *LoadContent()* function shown following. This function is fairly self-explanatory; it loads all the content or assets of your project, such as the game art and audio. It is important to note that this function is called only once per execution. Similarly, *UnloadContent()* is also called once per execution. However, as you may have guessed, it is called at the end of the game. It unloads all the content loaded by *LoadContent()*.

```
/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
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
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non-ContentManager content here
}
```

The next two functions, *Update()* and *Draw()*, behave fundamentally differently from *LoadContent()* and *UnloadContent()* in that they are both called many times per second throughout game execution. For example, it's not unusual for *Draw()* to be called 60 times per second and *Update()* to be called at an even higher rate.

The *Update()* function handles any change to the game state, and is also responsible for polling the input devices. This means that whenever the position of a game object within the game world is modified, or an input device (such as a controller) needs its state checked, that change gets passed through the *Update()* function. The following code shows the *Update()* function polling the state of a gamepad to see if it should exit the game.

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

    // TODO: Add your update logic here
    base.Update(gameTime);
}
```

The *Draw()* function draws the game world to the screen each time it is called, which is often around 60 times per second. *Draw()* clears the screen every cycle and redraws all the rendered game objects. In the example code for this game, the screen is cleared with cornflower blue at every draw cycle.



Note It is true that in general the *Draw()* function is called at a rate that is similar to the refresh frequency of your display monitor. However, these two rates are completely independent from each other. For example, changing your monitor's refresh rate has no effect on how often the *Draw()* function is called.

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

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
```

It is important to note that the purposes and operations of *Update()* and *Draw()* are well defined and distinct. The *Update()* function should update the state of the game (typically by changing instance variables) and should never draw anything, while the *Draw()* function should only display the game state (typically by rendering the relevant instance variables) and never change the game state. Mixing these two functions will cause many problems.

Adding, drawing, and controlling content

Now that you have a basic understanding of some of the functionality provided by *Xna.Framework*, let's go through the process of drawing and controlling something on the screen.

The Draw and Control project

This project demonstrates how to draw image files as textures and manipulate their positions. Figure 2-7 shows an example of what the project looks like when running.

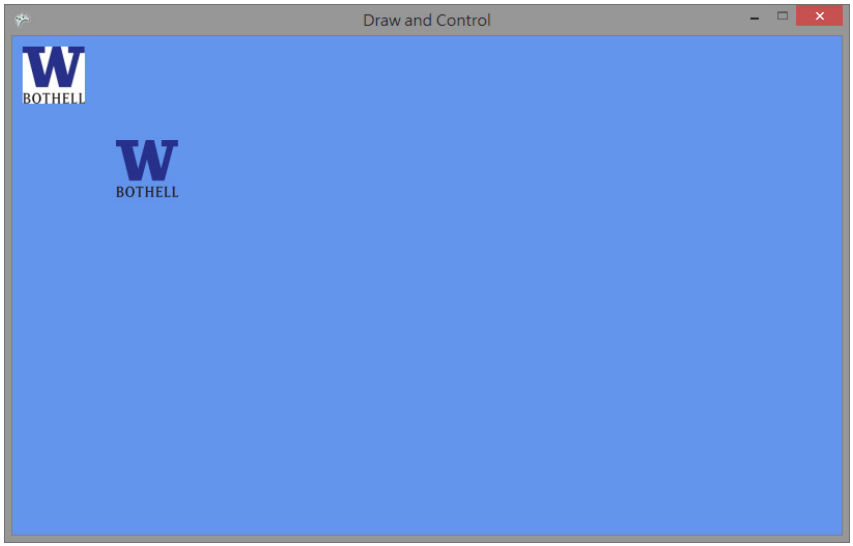


FIGURE 2-7 The Draw and Control project when it is running

The project's controls are as follows:

- **Back button (on the controller)** Exits the program
- **Left thumbstick** Moves the image with the white background
- **Right thumbstick** Moves the image with the transparent background

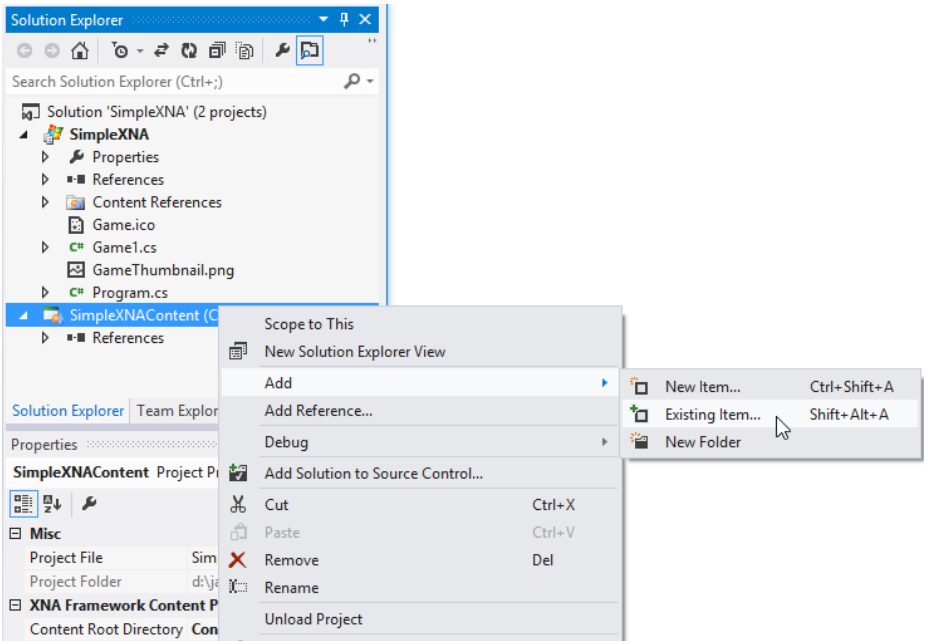
The goals of the project are as follows:

- To understand and experience working with the ContentProject and external images
- To understand the differences between .jpg and .png images (the transparency)
- To work with simple game states (position of images)
- To experience the basic drawing coordinate system and the *Update/Draw* loop

Creating the Draw and Control project

1. Start by creating a new project and naming it DrawAndControl. If you need to, you can refer back to the "Visual Studio development environment" section for a more detailed procedure.
2. Next, add the images to the content project in your game. Add a .jpg and a .png file to the project. This project uses UWB.jpg and UWB.png, which you can find in the sample code, in the Chapter02\ImagesUsed folder.

To add an image, right-click the DrawAndControlContent project and select Add | Existing Item, as shown in the following image. Alternatively, you can drag images from your file system and drop them onto the DrawAndControlContent project. Images you add to the project will be compiled into XNB format when the project is built. The XNB format is a binary file that XNA creates for the content pipeline. The compiled images will then be deployed as a part of the game.





Note The .png format supports transparency, or alpha, while the .jpg format does not. Transparency support for an image format means that areas of the image can be clear or partially clear. When an image has transparent parts, it uses the background to fill in those areas. When running the Draw and Control project, as shown in Figure 2-7, the difference between having and not having transparency support is easily recognizable.

3. With the images in your project, you can now define the game state. Do this by inserting the following lines of code into your *Game1* class.

   a. Declare the following instance variables: *Texture2D mJPGImage*, *Vector2 mJPGPosition*, *Texture2D mPNGImage*, and *Vector2 mPNGPosition*, as shown in the following code:

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager mGraphics;
    SpriteBatch mSpriteBatch;
```

```
    // Support for loading and drawing the JPG image
    Texture2D mJPGImage;     // The UWB-JPG.jpg image to be loaded
    Vector2 mJPGPosition;    // Top-Left pixel position of UWB-JPG.jpg

    // Support for loading and drawing of the PNG image
    Texture2D mPNGImage;     // The UWB-PNG.png image to be loaded
    Vector2 mPNGPosition;    // Top-Left pixel position of UWB-PNG.png
    ...
}
```

b. Next, initialize the game state by setting your images' starting position within the *Initialize()* function and loading your external images in the *LoadContent()* function:

```
protected override void Initialize()
{
    // Initialize the initial image positions.
    mJPGPosition = new Vector2(10f, 10f);
    mPNGPosition = new Vector2(100f, 100f);

    // Important to let the base class perform its initialization
    base.Initialize();
}

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

    // Load the images.
    mJPGImage = Content.Load<Texture2D>("UWB-JPG");
    mPNGImage = Content.Load<Texture2D>("UWB-PNG");
}
```

4. With the images loaded and ready, you can now draw them to your game world. You do this with the provided *SpriteBatch* object. As shown in the following code, the *SpriteBatch.Begin()* call initializes the drawing process, and the two *SpriteBatch.Draw()* calls add images to the batch to be rendered. When all desired drawing is done, you flush and display the drawn images with a call to the *SpriteBatch.End()* function. Notice that in the *SpriteBatch.End()* function, you simply draw the state of the game by rendering the images at the corresponding positions. You should avoid changing the game state information in the *Draw()* function.



Important Changes to the game state belong in the *Update()* function, with a few exceptions. Mixing the two functions will cause trouble because their calling frequencies are independent from each other.

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

    mSpriteBatch.Begin(); // Initialize drawing support
```

```
        // Draw the JPGImage
        mSpriteBatch.Draw(mJPGImage, mJPGPosition, Color.White);

        // Draw the PNGImage
        mSpriteBatch.Draw(mPNGImage, mPNGPosition, Color.White);

        mSpriteBatch.End(); // Inform graphics system we are done drawing

        base.Draw(gameTime);
    }
```

5. The last step is to update the position of the images based on the input from the user. As you can see in the following code, at every update you add to the images' current position by the gamepad's thumbstick displacement values. In this case, notice that the *Update()* function only updates the game state by changing the various instance variables. You should avoid drawing any of the game objects in the *Update()* function. Additionally, if you prefer to use the mouse and keyboard, code for this is also provided. Both the keyboard and mouse are handled by polling the current state for the key or button of interest and modifying the appropriate values. From this point on, projects will only use the game controller for input. If you wish to use the keyboard and mouse, you will need to add the code to your project.

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

        // Update the image positions with left/right thumbsticks
        mJPGPosition += GamePad.GetState(PlayerIndex.One).ThumbSticks.Left;
        mPNGPosition += GamePad.GetState(PlayerIndex.One).ThumbSticks.Right;
        #endregion

        #region Keyboard
        // Allows the game to exit
        if (Keyboard.GetState().IsKeyDown(Keys.Escape))
            this.Exit();

        // Update the image positions with Arrow keys
        if (Keyboard.GetState().IsKeyDown(Keys.Left))
            mJPGPosition.X--;
        if (Keyboard.GetState().IsKeyDown(Keys.Right))
            mJPGPosition.X++;
        if (Keyboard.GetState().IsKeyDown(Keys.Up))
            mJPGPosition.Y--;
        if (Keyboard.GetState().IsKeyDown(Keys.Down))
            mJPGPosition.Y++;

        // Update the image positions with AWSD
        if (Keyboard.GetState().IsKeyDown(Keys.A))
            mPNGPosition.X--;
        if (Keyboard.GetState().IsKeyDown(Keys.D))
            mPNGPosition.X++;
```

```
            if (Keyboard.GetState().IsKeyDown(Keys.W))
                mPNGPosition.Y--;
            if (Keyboard.GetState().IsKeyDown(Keys.S))
                mPNGPosition.Y++;
            #endregion

            #region Mouse
            // Poll mouse state
            MouseState mMouseState = Mouse.GetState();

            // If left mouse button is pressed
            if (mMouseState.LeftButton == ButtonState.Pressed)
                mJPGPosition = new Vector2(mMouseState.X, mMouseState.Y);

            // If right mouse button is pressed
            if (mMouseState.RightButton == ButtonState.Pressed)
                mPNGPosition = new Vector2(mMouseState.X, mMouseState.Y);
            #endregion

            base.Update(gameTime);
        }
```

The project is now ready. Build the project to see if any errors occur. If not, run the project and explore the program by moving the images around.



Note Remember to have your controller connected before executing the program.

Summary

This chapter has shown you how to create and explore a default generated XNA project. You saw how to customize your project by working with the *Initialize()*, *Load()*, *Draw()*, and *Update()* functions provided by the XNA framework. Using those four functions, you created a simple project that allows you to interact with your own graphical objects.

In general, the *Initialize()*, *Load()*, *Draw()*, and *Update()* functions provide you with the core functionality you need to build most games. These four functions afford you the ability to create games of limitless variety and complexity while also allowing you to work within the provided framework.

Lastly, you should pay special attention to the *Draw/Update* loop concept, because that concept spans nearly every technology and programming language as a practical and reliable way to build games. As you will see in the next chapter, you will increasingly use these functions in various ways as the projects become more complex.

Quick reference

To	Do this
Create a new XNA project	In Visual Studio, select File | New Project and choose Windows Game (4.0). Remember to name the project before you click OK.
Open the Solution Explorer tab	In Visual Studio, select View | Solution Explorer.
Change the game window title	In Solution Explorer, open the Properties tab, and in *AssemblyInfo.cs*, modify *[assembly: AssemblyTitle("YourTitle")]*.
Change the unique name of the project	In Solution Explorer, open the Properties tab, and in *AssemblyInfo.cs*, modify *[assembly: Guid("Your new GUID")]*.
Add source code to your game	Modify the Game1.cs source code file.
Include an image in your project	In Solution Explorer, right-click over the *ContentProject*, select Add | Existing Item, and browse to locate your image.
Load an image	In the Game1.cs source code file, define an instance variable for referencing to the image, like so: *Texture2D mTheImage;* Load the image into the variable in *LoadContent()* function, as follows: *mTheImage = Content.Load<Texture2D>("TheImageFileName");* In the Game1.cs source code file, define a variable to hold the image position, like so: *Vector2 mTheImagePosition;*
Draw the loaded image at a specific position	Draw the image in the *Draw()* function, as follows: *mSpriteBatch.Draw(mTheImage, mTheImagePosition, Color.White);*
Move the image	Update the image position with the gamepad's thumbstick movement in the *Update()* function, like so: *mTheImagePosition += GamePad.GetState(PlayerIndex.One). ThumbSticks.Left;* Draw the image *Draw()* function, as previously: *mSpriteBatch.Draw(mTheImage, mTheImagePosition, Color.White);* Note: Do not attempt to draw in *Update()*, and do not attempt to change any instance variables in *Draw()*.

Microsoft Press Ebooks—Your bookshelf on your devices!



When you buy an ebook through oreilly.com you get lifetime access to the book, and whenever possible we provide it to you in five, DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, Android .apk, and DAISY—that you can use on the devices of your choice. Our ebook files are fully searchable, and you can cut-and-paste and print them. We also alert you when we've updated the files with corrections and additions.

Learn more at ebooks.oreilly.com

You can also purchase O'Reilly ebooks through the iBookstore, the Android Marketplace, and Amazon.com.

O'REILLY®

Spreading the knowledge of innovators

oreilly.com