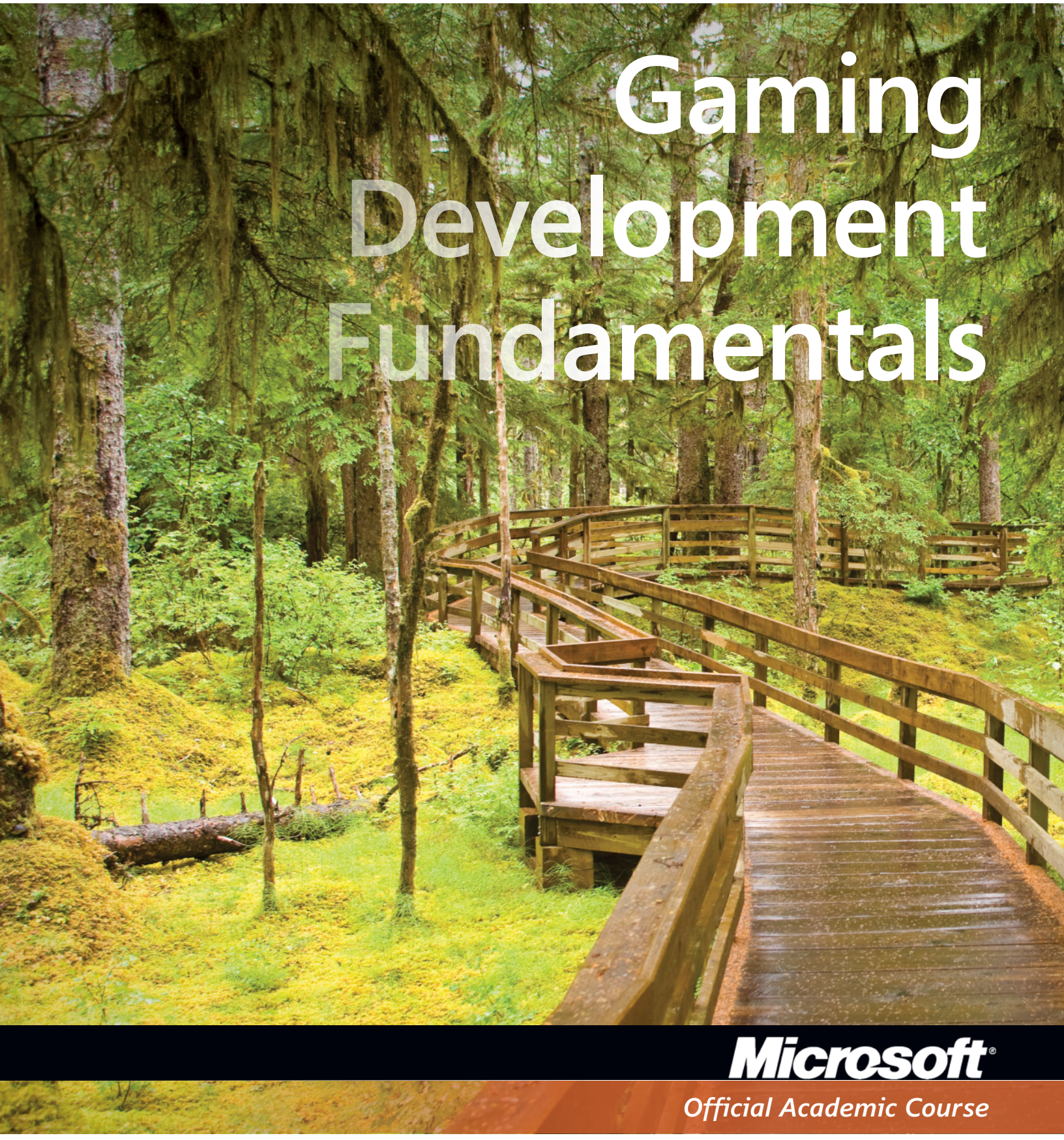


Gaming Development Fundamentals

Microsoft®

Official Academic Course

MTA Exam 98-374

This page is intentionally left blank

www.itbookshub.com

Microsoft® Official Academic Course

Gaming Development Fundamentals, Exam 98-374

WILEY

www.itbookshub.com

VP & PUBLISHER	Don Fowley
EDITOR	Bryan Gambrel
DIRECTOR OF SALES	Mitchell Beaton
EXECUTIVE MARKETING MANAGER	Chris Ruel
MICROSOFT PRODUCT MANAGER	Gene R. Longo of Microsoft Learning
ASSISTANT EDITOR	Jennifer Lartz
EDITORIAL ASSISTANT	Allison Winkle
ASSISTANT MARKETING MANAGER	Debbie Martin
SENIOR PRODUCTION & MANUFACTURING MANAGER	Janis Soo
ASSOCIATE PRODUCTION MANAGER	Joel Balbin
PRODUCTION EDITOR	Eugenia Lee
CREATIVE DIRECTOR	Harry Nolan
COVER DESIGNER	Georgina Smith
TECHNOLOGY AND MEDIA	Tom Kulesa/Wendy Ashenberg

Cover photo: © Michael Melford/Getty Images, Inc.

This book was set in Garamond by Aptara, Inc. and printed and bound by Bind-Rite Robbinsville.
The cover was printed by Bind-Rite Robbinsville.

Copyright © 2014 by John Wiley & Sons, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc. 222 Rosewood Drive, Danvers, MA 01923, website www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201)748-6011, fax (201)748-6008, website http://www.wiley.com/go/permissions.

Microsoft, ActiveX, Excel, InfoPath, Microsoft Press, MSDN, OneNote, Outlook, PivotChart, PivotTable, PowerPoint, SharePoint, SQL Server, Visio, Visual Basic, Visual C#, Visual Studio, Windows, Windows 7, Windows Mobile, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

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

The 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, John Wiley & Sons, Inc., Microsoft Corporation, nor their resellers or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

All photos in this book were printed with permission of the copyright owner. For all other third party photo provisions in the text, the copyright holders are indicated near the photo. The remaining photos were created by the authors of this textbook and printed with their permission.

ISBN 978-1-118-35989-1

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

www.itbookshub.com

Foreword from the Publisher

Wiley's publishing vision for the Microsoft Official Academic Course series is to provide students and instructors with the skills and knowledge they need to use Microsoft technology effectively in all aspects of their personal and professional lives. Quality instruction is required to help both educators and students get the most from Microsoft's software tools and to become more productive. Thus our mission is to make our instructional programs trusted educational companions for life.

To accomplish this mission, Wiley and Microsoft have partnered to develop the highest quality educational programs for Information Workers, IT Professionals, and Developers. Materials created by this partnership carry the brand name "Microsoft Official Academic Course," assuring instructors and students alike that the content of these textbooks is fully endorsed by Microsoft, and that they provide the highest quality information and instruction on Microsoft products. The Microsoft Official Academic Course textbooks are "Official" in still one more way—they are the officially sanctioned courseware for Microsoft IT Academy members.

The Microsoft Official Academic Course series focuses on *workforce development*. These programs are aimed at those students seeking to enter the workforce, change jobs, or embark on new careers as information workers, IT professionals, and developers. Microsoft Official Academic Course programs address their needs by emphasizing authentic workplace scenarios with an abundance of projects, exercises, cases, and assessments.

The Microsoft Official Academic Courses are mapped to Microsoft's extensive research and job-task analysis, the same research and analysis used to create the Microsoft Technology Associate (MTA) exams. The textbooks focus on real skills for real jobs. As students work through the projects and exercises in the textbooks they enhance their level of knowledge and their ability to apply the latest Microsoft technology to everyday tasks. These students also gain resume-building credentials that can assist them in finding a job, keeping their current job, or in furthering their education.

The concept of life-long learning is today an utmost necessity. Job roles, and even whole job categories, are changing so quickly that none of us can stay competitive and productive without continuously updating our skills and capabilities. The Microsoft Official Academic Course offerings, and their focus on Microsoft certification exam preparation, provide a means for people to acquire and effectively update their skills and knowledge. Wiley supports students in this endeavor through the development and distribution of these courses as Microsoft's official academic publisher.

Today educational publishing requires attention to providing quality print and robust electronic content. By integrating Microsoft Official Academic Course products, *WileyPLUS*, and Microsoft certifications, we are better able to deliver efficient learning solutions for students and teachers alike.

Joseph Heider

General Manager and Senior Vice President

www.itbookshub.com

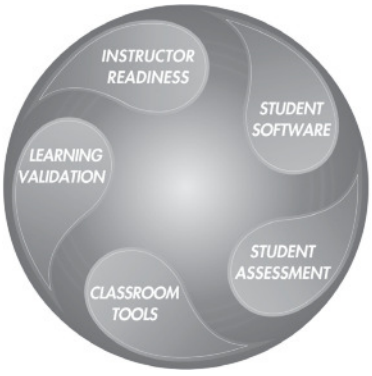
Preface

Welcome to the Microsoft Official Academic Course (MOAC) program for Gaming Development Fundamentals. MOAC represents the collaboration between Microsoft Learning and John Wiley & Sons, Inc. Microsoft and Wiley teamed up to produce a series of textbooks that deliver compelling and innovative teaching solutions to instructors and superior learning experiences for students. Infused and informed by in-depth knowledge from the creators of Microsoft products, and crafted by a publisher known worldwide for the pedagogical quality of its products, these textbooks maximize skills transfer in minimum time. Students are challenged to reach their potential by using their new technical skills as highly productive members of the workforce.

Because this knowledgebase comes directly from Microsoft, creator of the Microsoft Technology Associate (MTA) exams (www.microsoft.com/learning/certification), you are sure to receive the topical coverage that is most relevant to students' personal and professional success. Microsoft's direct participation not only assures you that MOAC textbook content is accurate and current; it also means that students will receive the best instruction possible to enable their success on certification exams and in the workplace.

The Microsoft Official Academic Course Program

The *Microsoft Official Academic Course* series is a complete program for instructors and institutions to prepare and deliver great courses on Microsoft software technologies. With MOAC, we recognize that, because of the rapid pace of change in the technology and curriculum developed by Microsoft, there is an ongoing set of needs beyond classroom instruction tools for an instructor to be ready to teach the course. The MOAC program endeavors to provide solutions for all these needs in a systematic manner in order to ensure a successful and rewarding course experience for both instructor and student—technical and curriculum training for instructor readiness with new software releases; the software itself for student use at home for building hands-on skills, assessment, and validation of skill development; and a great set of tools for delivering instruction in the classroom and lab. All are important to the smooth delivery of an interesting course on Microsoft software, and all are provided with the MOAC program. We think about the model below as a gauge for ensuring that we completely support you in your goal of teaching a great course. As you evaluate your instructional materials options, you may wish to use the model for comparison purposes with available products.



www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

www.itbookshub.com

Illustrated Book Tour

Pedagogical Features

The MOAC textbook for Gaming Development Fundamentals is designed to cover all the learning objectives for MTA Exam 98-374, which are referred to as its "objective domains." The Microsoft Technology Associate (MTA) exam objectives are highlighted throughout the textbook. Many pedagogical features have been developed specifically for *Microsoft Official Academic Course* programs.

Presenting the extensive procedural information and technical concepts woven throughout the textbook raises challenges for the student and instructor alike. The Illustrated Book Tour that follows provides a guide to the rich features contributing to the *Microsoft Official Academic Course* program's pedagogical plan. Following is a list of key features in each lesson designed to prepare students for success as they continue in their IT education, on the certification exams, and in the workplace:

- Each lesson begins with an **Exam Objective Matrix**. More than a standard list of learning objectives, the Exam Objective Matrix correlates each software skill covered in the lesson to the specific exam objective domain.
- Concise and frequent **Step-by-Step** instructions teach students new features and provide an opportunity for hands-on practice. Numbered steps give detailed, step-by-step instructions to help students learn software skills.
- **Illustrations:** Screen images provide visual feedback as students work through the exercises. The images reinforce key concepts, provide visual clues about the steps, and allow students to check their progress.
- **Key Terms:** Important technical vocabulary is listed with definitions at the beginning of the lesson. When these terms are used later in the lesson, they appear in bold italic type and are defined. The Glossary contains all of the key terms and their definitions.
- Engaging point-of-use **Reader Aids**, located throughout the lessons, tell students why this topic is relevant (*The Bottom Line*), and provide students with helpful hints (*Take Note*). Reader Aids also provide additional relevant or background information that adds value to the lesson.
- **Certification Ready** features throughout the text signal students where a specific certification objective is covered. They provide students with a chance to check their understanding of that particular MTA objective and, if necessary, review the section of the lesson where it is covered. MOAC offers complete preparation for MTA certification.
- **End-of-Lesson Questions:** The Knowledge Assessment section provides a variety of multiple-choice, true-false, matching, and fill-in-the-blank questions.
- **End-of-Lesson Exercises:** Competency Assessment case scenarios and Proficiency Assessment case scenarios are projects that test students' ability to apply what they've learned in the lesson.

www.itbookshub.com

■ Lesson Features

4 LESSON

Designing Specific Game Components

EXAM OBJECTIVE MATRIX

Skills/Concepts	MTA Exam Objective	MTA Exam Objective Number
Designing Game States and Loops	Plan for game state.	3.2
Designing Objects and Characters	Animate basic characters. Transform objects.	4.1 4.2
Designing Physics-Based Animations	Work with collisions.	4.3

KEY TERMS

anisotropic filtering
collision detection
collision response
filter
fixed step game loop
frames per second (fps)
game loops
gameflow
graphics pipeline
general-purpose computing on graphics processing unit (GPGPU)
High Level Shading Language (HLSL)
interpolation
linear filtering
mipmap
nonplayer characters (NPC)
per-pixel lighting
physics engine
physics processing unit (PPU)
physics simulation
pixel shader
point filtering
projection matrix
projection space
scene hierarchy
scripted-events
shader
sprite animation
texels
texture mapping
variable step game loop
vertex shader
view matrix
view space
world matrix
world space

84

Exam Objective Matrix

Key Terms

Business Scenario

Steve Watson works at Contoso Gaming Inc., and leads the development team. His company is developing a first-person shooter (FPS) game for console and computer systems. The team has designed the game's visual world. They are now in the process of designing various components for their game using XNA 4.0.

As a first step, Steve and his team decide to provide a good gameflow for their game. For this, they identify a well-designed sequence of challenges and rewards for their game in order to move the game story forward. They sequence the challenges in such a way that it increases the game's complexity at each level successively. They also decide to utilize every element of the game efficiently through optimizing the states of their game. This ensures that their game runs smoothly across different platforms. The central component of any game design is game loop. Steve's team implements the main loop in XNA to help the game to run smoothly irrespective of the player's input.

Players interact with a game through objects and characters. To make these objects and characters come alive on the screen, Steve's team adds the minutest detail to these game elements, such as the types of movement they can make, the tasks they can perform, and so on. Moreover, the team also decides to simulate artificial intelligence to their game characters to help retain the interest of players.

■ Designing Game States and Loops

THE BOTTOM LINE

You should design beforehand the gameflow—and the properties of the objects and characters required at each game state in the gameflow—as well as the actions the characters can perform. It is also important to understand the scope of artificial intelligence (AI) in your game so that the end product lives up to the developer's and player's expectation.

The design of the game does not end with creating the visual design. You need to design the progress of the game from one game state to the next. You also need to decide which characters and objects will be present at each game state and the actions they will be able to perform. In addition, you need to decide whether your game or the characters in the game are going to have built-in intelligence or AI.

The game components that you need to design at this stage include:

- Gameflow, game states, and game loops
- Objects and characters
- Physics-based animations and AI

Creating Gameflow

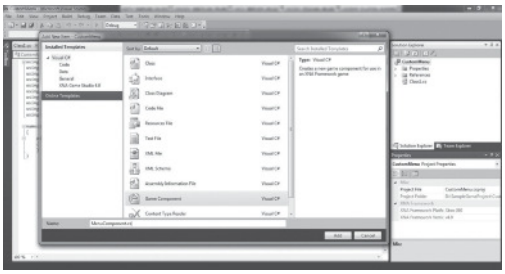
Gameflow is the progression of the game from one state to another state. It comprises of a sequence of challenging tasks and provides rewards to players to motivate completion. Gameflow makes players experience the game and gives them a sense of accomplishment.

CERTIFICATION READY
What are the different aspects to a gameflow?
3.2

Video games often have a series of challenging tasks or complex situations. The goal of the game is for the players to complete the tasks and solve the situations to receive rewards. The reward can be anything—moving to a higher game level, another life, some booty, new weapons, access to specific areas of the game, some extra points, and so on. These rewards

Bottom Line

Figure 5-4
The Add New Item dialog box listing GameComponent



Screen Images

```
        public Vector2 position;
       /* defines the normal color of the menu item */
        public Color normal = Color.White;
       /* defines the color of the menu item when it is
highlighted */
        public Color highlight = Color.Yellow;
       /* constructor that initializes the name and the posi-
tion of the menuitem */
        public MenuItem(string name,Vector2 pos)
        {
            menuName = name;
            position = pos;
        }
    }
}
```

2. Write the following code in the MenuComponent class:

```
namespace CustomMenu
{
public class MenuComponent : Microsoft.Xna.Framework.
DrawableGameComponent
    {
        /* holds all menu items */
        public List<MenuItem> allButtons;
      /* holds the index of the clicked menu item*/
       public int clickedButtonIndex;
      /* holds the current keyboard state */
```

www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

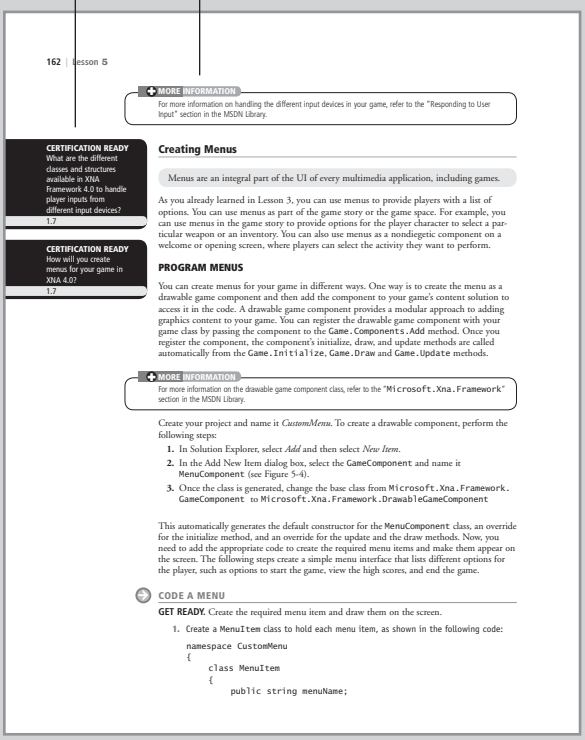
www.itbookshub.com

Certification Ready Alert

More Information Reader Aid

Take Note Reader Aid

Cross Reference Reader Aid



162 | Lesson 5

MORE INFORMATION

For more information on handling the different input devices in your game, refer to the "Responding to User Input" section in the MSDN Library.

CERTIFICATION READY
What are the different classes and structures available in XNA Framework 4.0 to handle player inputs from different input devices?
1.7

CERTIFICATION READY
How will you create menus for your game in XNA 4.0?
1.7

Creating Menus

Menus are an integral part of the UI of every multimedia application, including games.

As you already learned in Lesson 3, you can use menus to provide players with a list of options. You can use menus as part of the game story or the game space. For example, you can use menus in the game story to provide options for the player character to select a particular weapon or an inventory. You can also use menus as a nondiegetic component on a welcome or opening screen, where players can select the activity they want to perform.

PROGRAM MENUS

You can create menus for your game in different ways. One way is to create the menu as a drawable game component and then add the component to your game's content solution to access it in the code. A drawable game component provides a modular approach to adding graphics content to your game. You can register the drawable game component with your game class by passing the component to the `Game.Components.Add` method. Once you register the component, the component's initialize, draw, and update methods are called automatically from the `Game.Initialize`, `Game.Draw` and `Game.Update` methods.

MORE INFORMATION

For more information on the drawable game component class, refer to the "`Microsoft.Xna.Framework`" section in the MSDN Library.

Create your project and name it *CustomMenu*. To create a drawable component, perform the following steps:

1. In Solution Explorer, select *Add* and then select *New Item*.
2. In the Add New Item dialog box, select the `GameComponent` and name it `MenuComponent` (see Figure 5-4).
3. Once the class is generated, change the base class from `Microsoft.Xna.Framework.GameComponent` to `Microsoft.Xna.Framework.DrawableGameComponent`

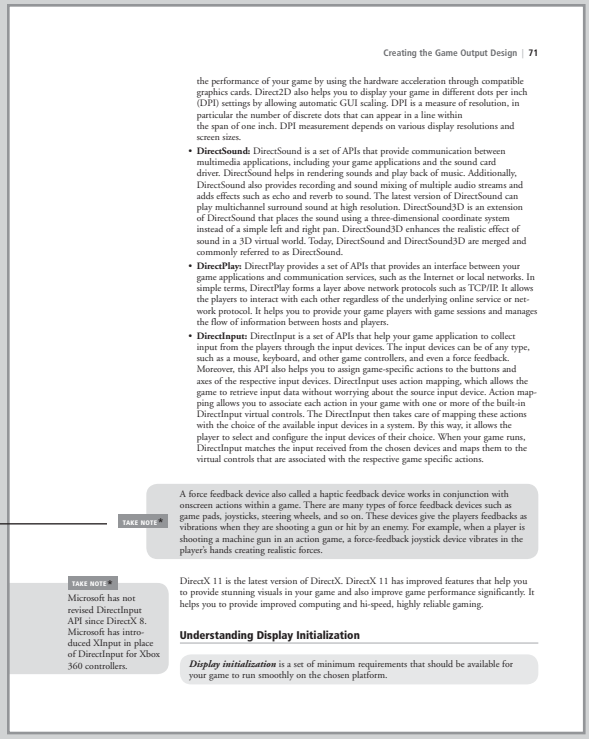
This automatically generates the default constructor for the `MenuComponent` class, an override for the initialize method, and an override for the update and the draw methods. Now, you need to add the appropriate code to create the required menu items and make them appear on the screen. The following steps create a simple menu interface that lists different options for the player, such as options to start the game, view the high scores, and end the game.

CODE A MENU

GET READY. Create the required menu item and draw them on the screen.

1. Create a `MenuItem` class to hold each menu item, as shown in the following code:

```
namespace CustomMenu
{
    class MenuItem
    {
        public string menuName;
```



the performance of your game by using the hardware acceleration through compatible graphics cards. Direct2D also helps you to display your game in different dots per inch (DPI) settings by allowing automatic GUI scaling. DPI is a measure of resolution, in particular the number of discrete dots that can appear in a line within the span of one inch. DPI measurement depends on various display resolutions and screen sizes.

- **DirectSound:** DirectSound is a set of APIs that provide communication between multimedia applications, including your game applications and the sound card driver. DirectSound helps in rendering sounds and play back of music. Additionally, DirectSound also provides recording and sound mixing of multiple audio streams and adds effects such as echo and reverb to sound. The latest version of DirectSound can play multichannel surround sound at high resolution. DirectSound3D is an extension of DirectSound that places the sound using a three-dimensional coordinate system instead of a simple left and right pan. DirectSound3D enhances the realistic effect of sound in a 3D virtual world. Today, DirectSound and DirectSound3D are merged and commonly referred to as DirectSound.
- **DirectPlay:** DirectPlay provides a set of APIs that provides an interface between your game applications and communication services, such as the Internet or local networks. In simple terms, DirectPlay forms a layer above network protocols such as TCP/IP. It allows the players to interact with each other regardless of the underlying online service or network protocol. It helps you to provide your game players with game sessions and manages the flow of information between hosts and players.
- **DirectInput:** DirectInput is a set of APIs that help your game application to collect input from the players through the input devices. The input devices can be of any type, such as a mouse, keyboard, and other game controllers, and even a force feedback. Moreover, this API also helps you to assign game-specific actions to the buttons and axes of the respective input devices. DirectInput uses action mapping, which allows the game to retrieve input data without worrying about the source input device. Action mapping allows you to associate each action in your game with one or more of the built-in DirectInput virtual controls. The DirectInput then takes care of mapping these actions with the choice of the available input devices in a system. By this way, it allows the player to select and configure the input devices of their choice. When your game runs, DirectInput matches the input received from the chosen devices and maps them to the virtual controls that are associated with the respective game specific actions.

TAKE NOTE*

A force feedback device also called a haptic feedback device works in conjunction with onscreen actions within a game. There are many types of force feedback devices such as game pads, joysticks, steering wheels, and so on. These devices give the players feedbacks as vibrations when they are shooting a gun or hit by an enemy. For example, when a player is shooting a machine gun in an action game, a force-feedback joystick device vibrates in the player's hands creating realistic forces.

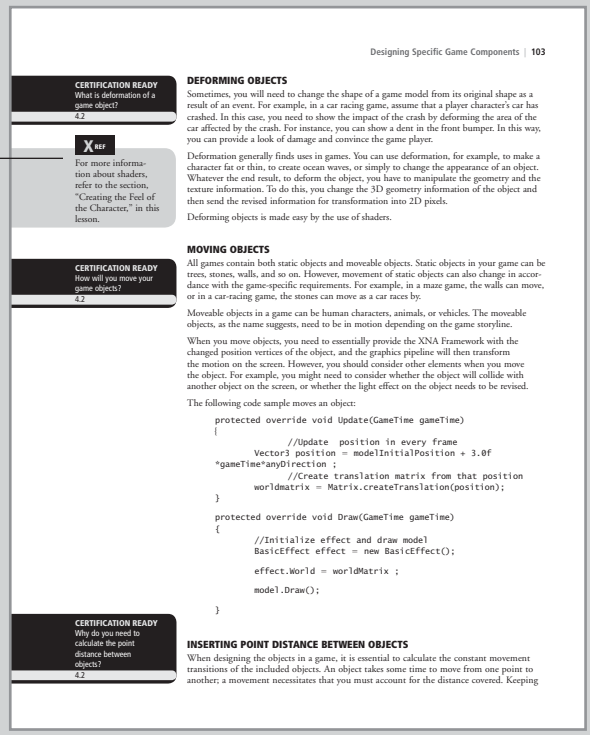
TAKE NOTE*

Microsoft has not revised DirectInput API since DirectX 8. Microsoft has introduced XInput in place of DirectInput for Xbox 360 controllers.

DirectX 11 is the latest version of DirectX. DirectX 11 has improved features that help you to provide stunning visuals in your game and also improve game performance significantly. It helps you to provide improved computing and hi-speed, highly reliable gaming.

Understanding Display Initialization

Display initialization is a set of minimum requirements that should be available for your game to run smoothly on the chosen platform.



CERTIFICATION READY
What is deformation of a game object?
4.2

XREF

For more information about shaders, refer to the section, "Creating the Feel of the Character," in this lesson.

DEFORMING OBJECTS

Sometimes, you will need to change the shape of a game model from its original shape as a result of an event. For example, in a car racing game, assume that a player character's car has crashed. In this case, you need to show the impact of the crash by deforming the area of the car affected by the crash. For instance, you can show a dent in the front bumper. In this way, you can provide a look of damage and convince the game player.

Deformation generally finds uses in games. You can use deformation, for example, to make a character fat or thin, to create ocean waves, or simply to change the appearance of an object. Whatever the end result, to deform the object, you have to manipulate the geometry and the texture information. To do this, you change the 3D geometry information of the object and then send the revised information for transformation into 2D pixels.

Deforming objects is made easy by the use of shaders.

CERTIFICATION READY
How will you move your game objects?
4.2

MOVING OBJECTS

All games contain both static objects and moveable objects. Static objects in your game can be trees, stones, walls, and so on. However, movement of static objects can also change in accordance with the game-specific requirements. For example, in a maze game, the walls can move, or in a car-racing game, the stones can move as a car races by.

Moveable objects in a game can be human characters, animals, or vehicles. The moveable objects, as the name suggests, need to be in motion depending on the game storyline.

When you move objects, you need to essentially provide the XNA Framework with the changed position vertices of the object, and the graphics pipeline will then transform the motion on the screen. However, you should consider other elements when you move the object. For example, you might need to consider whether the object will collide with another object on the screen, or whether the light effect on the object needs to be revised.

The following code sample moves an object:

```
protected override void Update(GameTime gameTime)
{
        //Update  position in every frame
    Vector3 position = modelInitialPosition + 3.0f
*gameTime*anyDirection ;
        //Create translation matrix from that position
    worldmatrix = Matrix.createTranslation(position);
}

protected override void Draw(GameTime gameTime)
{
    //Initialize effect and draw model
    BasicEffect effect = new BasicEffect();

    effect.World = worldMatrix ;

    model.Draw();

}
```

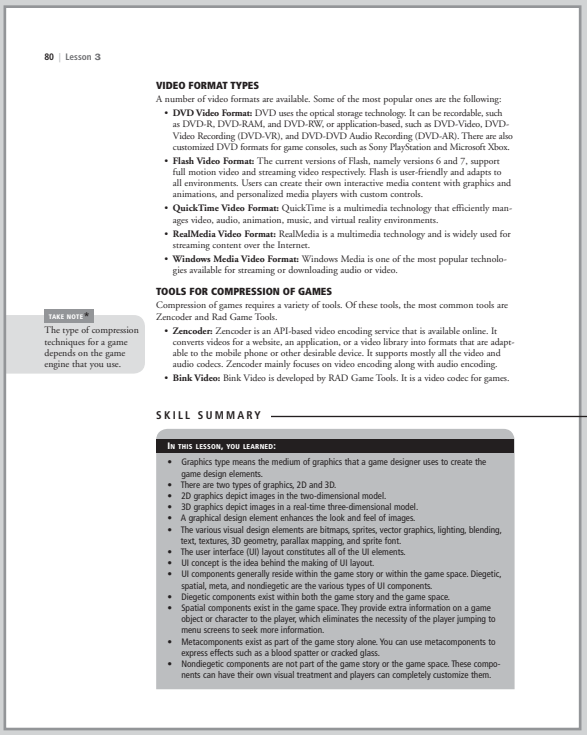
CERTIFICATION READY
Why do you need to calculate the point distance between objects?
4.2

INSERTING POINT DISTANCE BETWEEN OBJECTS

When designing the objects in a game, it is essential to calculate the constant movement transitions of the included objects. An object takes some time to move from one point to another; a movement necessitates that you must account for the distance covered. Keeping

www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

www.itbookshub.com



80 | Lesson 3

VIDEO FORMAT TYPES

A number of video formats are available. Some of the most popular ones are the following:

- **DVD Video Format:** DVD uses the optical storage technology. It can be recordable, such as DVD-R, DVD-RAM, and DVD-RW, or application-based, such as DVD-Video, DVD-Video Recording (DVD-VR), and DVD-DVD Audio Recording (DVD-AR). There are also customized DVD formats for game consoles, such as Sony PlayStation and Microsoft Xbox.
- **Flash Video Format:** The current versions of Flash, namely versions 6 and 7, support full motion video and streaming video respectively. Flash is user-friendly and adapts to all environments. Users can create their own interactive media content with graphics and animations, and personalized media players with custom controls.
- **QuickTime Video Format:** QuickTime is a multimedia technology that efficiently manages video, audio, animation, music, and virtual reality environments.
- **RealMedia Video Format:** RealMedia is a multimedia technology and is widely used for streaming content over the Internet.
- **Windows Media Video Format:** Windows Media is one of the most popular technologies available for streaming or downloading audio or video.

TOOLS FOR COMPRESSION OF GAMES

Compression of games requires a variety of tools. Of these tools, the most common tools are Zencoder and Rad Game Tools.

TAKE NOTE

The type of compression techniques for a game depends on the game engine that you use.

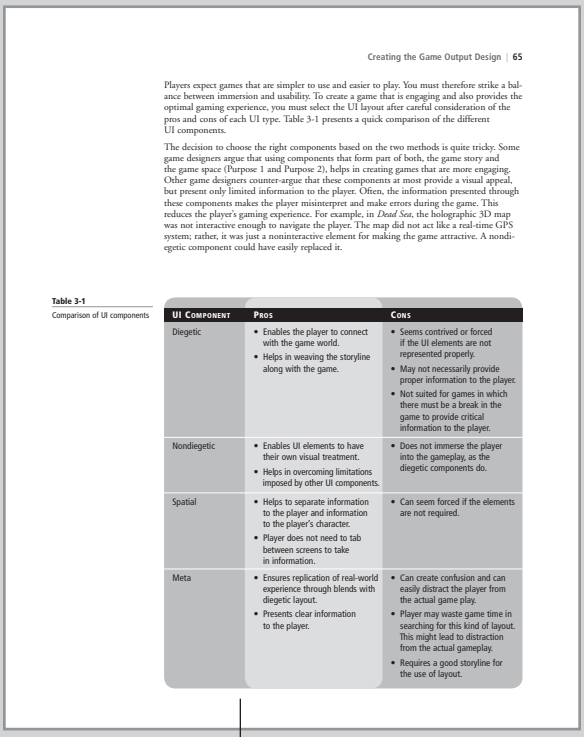
- **Zencoder:** Zencoder is an API-based video encoding service that is available online. It converts videos for a website, an application, or a video library into formats that are adaptable to the mobile phone or other desirable device. It supports mostly all the video and audio codecs. Zencoder mainly focuses on video encoding along with audio encoding.
- **Bink Video:** Bink Video is developed by RAD Game Tools. It is a video codec for games.

SKILL SUMMARY

IN THIS LESSON, YOU LEARNED:

- Graphics type means the medium of graphics that a game designer uses to create the game design elements.
- There are two types of graphics, 2D and 3D.
- 2D graphics depict images in the two-dimensional model.
- 3D graphics depict images in a real-time three-dimensional model.
- A graphical design element enhances the look and feel of images.
- The various visual design elements are bitmaps, sprites, vector graphics, lighting, blending, text, textures, 3D geometry, parallax mapping, and sprite font.
- The user interface (UI) layout constitutes all of the UI elements.
- UI concept is the idea behind the making of UI layout.
- UI components generally reside within the game story or within the game space. Diegetic, spatial, meta, and nondiegetic are the various types of UI components.
- Diegetic components exist within both the game story and the game space.
- Spatial components exist in the game space. They provide extra information on a game object or character to the player, which eliminates the necessity of the player jumping to menu screens to seek more information.
- Metacomponents exist as part of the game story alone. You can use metacomponents to express effects such as a blood spatter or cracked glass.
- Nondiegetic components are not part of the game story or the game space. These components can have their own visual treatment and players can completely customize them.

Skill Summary



Creating the Game Output Design | 65

Players expect games that are simpler to use and easier to play. You must therefore strike a balance between immersion and usability. To create a game that is engaging and also provides the optimal gaming experience, you must select the UI layout after careful consideration of the pros and cons of each UI type. Table 3-1 presents a quick comparison of the different UI components.

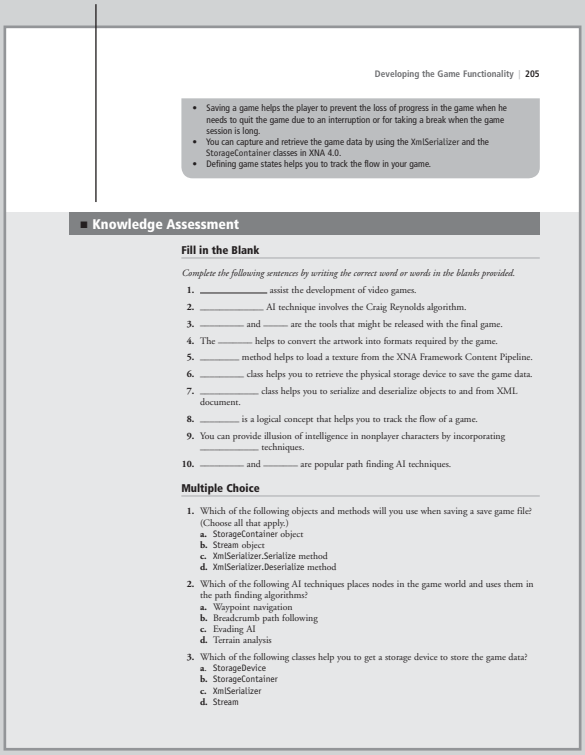
The decision to choose the right components based on the two methods is quite tricky. Some game designers argue that using components that form part of both, the game story and the game space (Purpose 1 and Purpose 2), helps in creating games that are more engaging. Other game designers counter-argue that these components at most provide a visual appeal, but present only limited information to the player. Often, the information presented through these components makes the player misinterpret and make errors during the game. This reduces the player's gaming experience. For example, in *Dead Sea*, the holographic 3D map was not interactive enough to navigate the player. The map did not act like a real-time GPS system; rather, it was just a noninteractive element for making the game attractive. A nondiegetic component could have easily replaced it.

Table 3-1
Comparison of UI components

UI COMPONENT	PROS	CONS
Diegetic	• Enables the player to connect with the game world. • Helps in weaving the storyline along with the game.	• Seems contrived or forced if the UI elements are not represented properly. • May not necessarily provide proper information to the player. • Not suited for games in which there must be a break in the game to provide critical information to the player.
Nondiegetic	• Enables UI elements to have their own visual treatment. • Helps in overcoming limitations imposed by other UI components.	• Does not immerse the player into the gameplay, as the diegetic components do.
Spatial	• Helps to separate information to the player and information to the player's character. • Player does not need to tab between screens to take in information.	• Can seem forced if the elements are not required.
Meta	• Ensures replication of real-world experience through blends with diegetic layout. • Presents clear information to the player.	• Can create confusion and can easily distract the player from the actual game play. • Player may waste game time in searching for this kind of layout. This might lead to distraction from the actual gameplay. • Requires a good storyline for the use of layout.

Easy-to-Read Tables

Knowledge Assessment Questions



Developing the Game Functionality | 205

- Saving a game helps the player to prevent the loss of progress in the game when he needs to quit the game due to an interruption or for taking a break when the game session is long.
- You can capture and retrieve the game data by using the XmlSerializer and the StorageContainer classes in XNA 4.0.
- Defining game states helps you to track the flow in your game.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

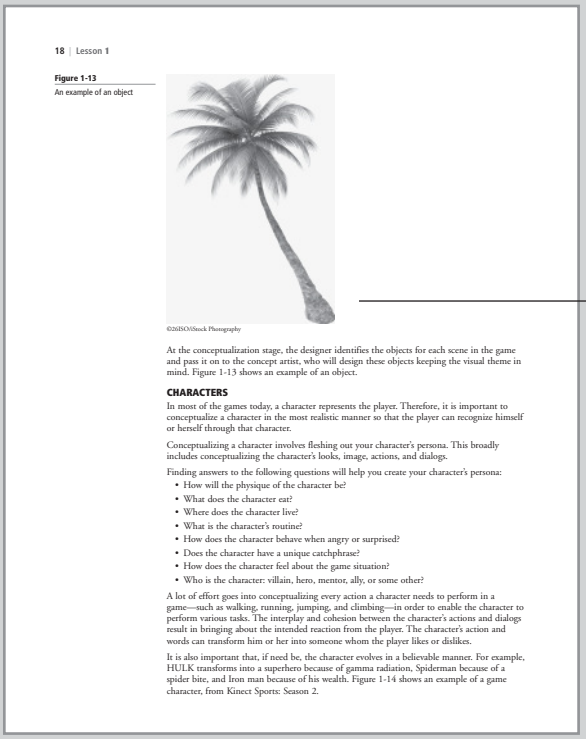
1. _____ assist the development of video games.
2. _____ AI technique involves the Craig Reynolds algorithm.
3. _____ and _____ are the tools that might be released with the final game.
4. The _____ helps to convert the artwork into formats required by the game.
5. _____ method helps to load a texture from the XNA Framework Content Pipeline.
6. _____ class helps you to retrieve the physical storage device to save the game data.
7. _____ class helps you to serialize and deserialize objects to and from XML document.
8. _____ is a logical concept that helps you to track the flow of a game.
9. You can provide illusion of intelligence in nonplayer characters by incorporating _____ techniques.
10. _____ and _____ are popular path finding AI techniques.

Multiple Choice

1. Which of the following objects and methods will you use when saving a save game file? (Choose all that apply.)
   a. StorageContainer object
   b. Stream object
   c. XmlSerializer.Serialize method
   d. XmlSerializer.Deserialize method
2. Which of the following AI techniques places nodes in the game world and uses them in the path finding algorithms?
   a. Waypoint navigation
   b. Breadcrumb path following
   c. Evading AI
   d. Terrain analysis
3. Which of the following classes help you to get a storage device to store the game data?
   a. StorageDevice
   b. StorageContainer
   c. XmlSerializer
   d. Stream

www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

www.itbookshub.com



18 | Lesson 1

Figure 1-13
An example of an object

©2615O/iStock Photography

At the conceptualization stage, the designer identifies the objects for each scene in the game and pass it on to the concept artist, who will design these objects keeping the visual theme in mind. Figure 1-13 shows an example of an object.

CHARACTERS

In most of the games today, a character represents the player. Therefore, it is important to conceptualize a character in the most realistic manner so that the player can recognize himself or herself through that character.

Conceptualizing a character involves fleshing out your character's persona. This broadly includes conceptualizing the character's looks, image, actions, and dialogs.

Finding answers to the following questions will help you create your character's persona:

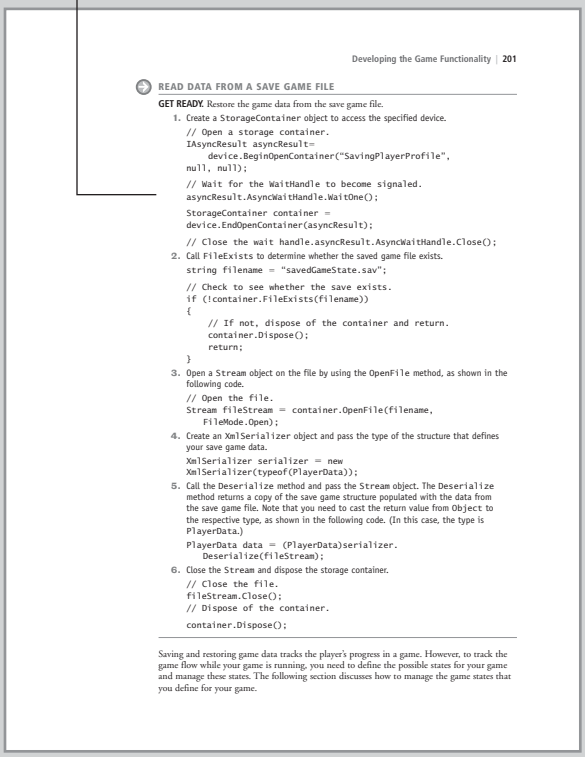
- How will the physique of the character be?
- What does the character eat?
- Where does the character live?
- What is the character's routine?
- How does the character behave when angry or surprised?
- Does the character have a unique catchphrase?
- How does the character feel about the game situation?
- Who is the character: villain, hero, mentor, ally, or some other?

A lot of effort goes into conceptualizing every action a character needs to perform in a game—such as walking, running, jumping, and climbing—in order to enable the character to perform various tasks. The interplay and cohesion between the character's actions and dialogs result in bringing about the intended reaction from the player. The character's action and words can transform him or her into someone whom the player likes or dislikes.

It is also important that, if need be, the character evolves in a believable manner. For example, HULK transforms into a superhero because of gamma radiation, Spiderman because of a spider bite, and Iron man because of his wealth. Figure 1-14 shows an example of a game character, from Kinect Sports: Season 2.

Photos

Step-by-Step Exercises



Developing the Game Functionality | 201

READ DATA FROM A SAVE GAME FILE

GET READY. Restore the game data from the save game file.

1. Create a `StorageContainer` object to access the specified device.

```
// Open a storage container.
IAsyncResult asyncResult=
    device.BeginOpenContainer("SavingPlayerProfile",
null, null);

// Wait for the WaitHandle to become signaled.
asyncResult.AsyncWaitHandle.WaitOne();

StorageContainer container =
device.EndOpenContainer(asyncResult);

// Close the wait handle.asyncResult.AsyncWaitHandle.Close();
```

2. Call `FileExists` to determine whether the saved game file exists.

```
string filename = "savedGameState.sav";

// Check to see whether the save exists.
if (!container.FileExists(filename))
{
    // If not, dispose of the container and return.
    container.Dispose();
    return;
}
```

3. Open a `Stream` object on the file by using the `OpenFile` method, as shown in the following code.

```
// Open the file.
Stream fileStream = container.OpenFile(filename,
    FileMode.Open);
```

4. Create an `XmlSerializer` object and pass the type of the structure that defines your save game data.

```
XmlSerializer serializer = new
XmlSerializer(typeof(PlayerData));
```

5. Call the `Deserialize` method and pass the `Stream` object. The `Deserialize` method returns a copy of the save game structure populated with the data from the save game file. Note that you need to cast the return value from `Object` to the respective type, as shown in the following code. (In this case, the type is `PlayerData`.)

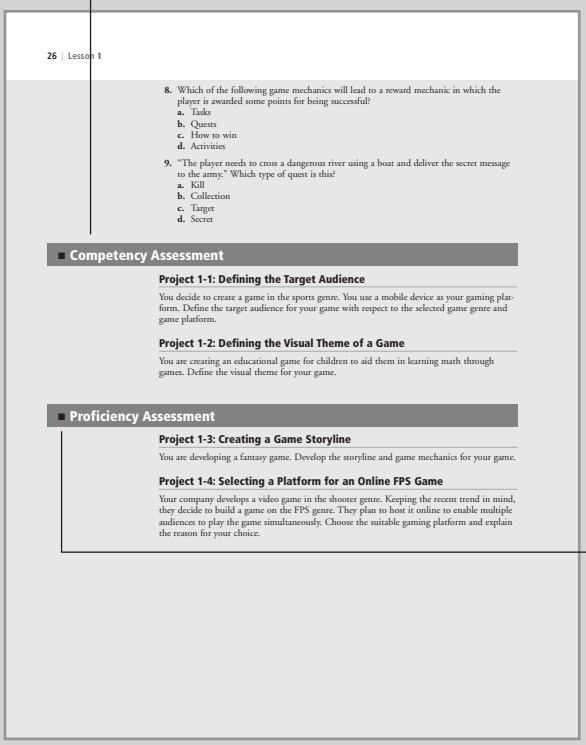
```
PlayerData data = (PlayerData)serializer.
    Deserialize(fileStream);
```

6. Close the `Stream` and dispose the storage container.

```
// Close the file.
fileStream.Close();
// Dispose of the container.

container.Dispose();
```

Saving and restoring game data tracks the player's progress in a game. However, to track the game flow while your game is running, you need to define the possible states for your game and manage these states. The following section discusses how to manage the game states that you define for your game.

Competency Assessment



26 | Lesson 1

8. Which of the following game mechanics will lead to a reward mechanic in which the player is awarded some points for being successful?
   a. Tasks
   b. Quests
   c. How to win
   d. Activities
9. "The player needs to cross a dangerous river using a boat and deliver the secret message to the army." Which type of quest is this?
   a. Kill
   b. Collection
   c. Target
   d. Secret

■ Competency Assessment

Project 1-1: Defining the Target Audience

You decide to create a game in the sports genre. You use a mobile device as your gaming platform. Define the target audience for your game with respect to the selected game genre and game platform.

Project 1-2: Defining the Visual Theme of a Game

You are creating an educational game for children to aid them in learning math through games. Define the visual theme for your game.

■ Proficiency Assessment

Project 1-3: Creating a Game Storyline

You are developing a fantasy game. Develop the storyline and game mechanics for your game.

Project 1-4: Selecting a Platform for an Online FPS Game

Your company develops a video game in the shooter genre. Keeping the recent trend in mind, they decide to build a game on the FPS genre. They plan to host it online to enable multiple audiences to play the game simultaneously. Choose the suitable gaming platform and explain the reason for your choice.

Proficiency Assessment

www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

Conventions and Features Used in This Book

This book uses particular fonts, symbols, and heading conventions to highlight important information or to call your attention to special steps. For more information about the features in each lesson, refer to the Illustrated Book Tour section.

Convention	Meaning
THE BOTTOM LINE	This feature provides a brief summary of the material to be covered in the section that follows.
CLOSE	Words in all capital letters indicate instructions for opening, saving, or closing files or programs. They also point out items you should check or actions you should take.
CERTIFICATION READY	This feature signals the point in the text where a specific certification objective is covered. It provides you with a chance to check your understanding of that particular MTA objective and, if necessary, review the section of the lesson where it is covered.
TAKE NOTE*	Reader aids appear in shaded boxes found in your text. *Take Note* provides helpful hints related to particular tasks or topics.
XREF	These notes provide pointers to information discussed elsewhere in the textbook or describe interesting gaming development features that are not directly addressed in the current topic or exercise.
Alt + Tab	A plus sign (+) between two key names means that you must press both keys at the same time. Keys that you are instructed to press in an exercise will appear in the font shown here.
Example	Key terms appear in bold italic.

Instructor Support Program

The *Microsoft Official Academic Course* programs are accompanied by a rich array of resources that incorporate the extensive textbook visuals to form a pedagogically cohesive package. These resources provide all the materials instructors need to deploy and deliver their courses. Resources available online for download include:

- **DreamSpark Premium** is designed to provide the easiest and most inexpensive developer tools, products, and technologies available to faculty and students in labs, classrooms, and on student PCs. A free 3-year membership is available to qualified MOAC adopters.

  Note: Microsoft Visual Studio, XNA Game Studio, and Windows Operating Systems can be downloaded from DreamSpark Premium for use by students in this course.
- The **Instructor Guide** contains Solutions to all the textbook exercises and Syllabi for various term lengths. The Instructor Guide also includes chapter summaries and lecture notes. The Instructor's Guide is available from the Book Companion site (http://www.wiley.com/college/microsoft).
- The **Test Bank** contains hundreds of questions in multiple-choice, true-false, short answer, and essay formats, and is available to download from the Instructor's Book Companion site (www.wiley.com/college/microsoft). A complete answer key is provided.
- A complete set of **PowerPoint presentations and images** is available on the Instructor's Book Companion site (http://www.wiley.com/college/microsoft) to enhance classroom presentations. Approximately 50 PowerPoint slides are provided for each lesson. Tailored to the text's topical coverage and Skills Matrix, these presentations are designed to convey key concepts addressed in the text. All images from the text are on the Instructor's Book Companion site (http://www.wiley.com/college/microsoft). You can incorporate them into your PowerPoint presentations, or create your own overhead transparencies and handouts. By using these visuals in class discussions, you can help focus students' attention on key elements of technologies covered and help them understand how to use it effectively in the workplace.

Wiley Faculty Network

- When it comes to improving the classroom experience, there is no better source of ideas and inspiration than your fellow colleagues. The **Wiley Faculty Network** connects teachers with technology, facilitates the exchange of best practices, and helps to enhance instructional efficiency and effectiveness. Faculty Network activities include technology training and tutorials, virtual seminars, peer-to-peer exchanges of experiences and ideas, personal consulting, and sharing of resources. For details visit www.WhereFacultyConnect.com.

DREAMSPARK PREMIUM—FREE 3-YEAR MEMBERSHIP AVAILABLE TO QUALIFIED ADOPTERS!

DreamSpark Premium is designed to provide the easiest and most inexpensive way for universities to make the latest Microsoft developer tools, products, and technologies available in labs, classrooms, and on student PCs. DreamSpark Premium is an annual membership program for departments teaching Science, Technology, Engineering, and Mathematics (STEM) courses. The membership provides a complete solution to keep academic labs, faculty, and students on the leading edge of technology.

Software available in the DreamSpark Premium program is provided at no charge to adopting departments through the Wiley and Microsoft publishing partnership.

Contact your Wiley rep for details.

For more information about the DreamSpark Premium program, go to:

https://www.dreamspark.com/

Note: Microsoft Visual Studio, XNA Game Studio, and Windows Operating Systems can be downloaded from DreamSpark Premium for use by students in this course.

■ Important Web Addresses and Phone Numbers

To locate the Wiley Higher Education Rep in your area, go to http://www.wiley.com/college and click on the "*Who's My Rep?*" link at the top of the page, or call the MOAC Toll Free Number: 1 + (888) 764-7001 (U.S. & Canada only).

To learn more about becoming certified and exam availability, visit www.microsoft.com/learning/mcp/mcp.

Additional Resources

Book Companion Website (www.wiley.com/college/microsoft)

The students' book companion site for the MOAC series includes any resources, exercise files, and Web links that will be used in conjunction with this course.

Wiley Desktop Editions

Wiley MOAC Desktop Editions are innovative, electronic versions of printed textbooks. Students buy the desktop version for up to 40% off the U.S. price of the printed text, and get the added value of permanence and portability. Wiley Desktop Editions provide students with numerous additional benefits that are not available with other e-text solutions.

Wiley Desktop Editions are NOT subscriptions; students download the Wiley Desktop Edition to their computer desktops. Students own the content they buy to keep for as long as they want. Once a Wiley Desktop Edition is downloaded to the computer desktop, students have instant access to all of the content without being online. Students can also print out the sections they prefer to read in hard copy. Students also have access to fully integrated resources within their Wiley Desktop Edition. From highlighting their e-text to taking and sharing notes, students can easily personalize their Wiley Desktop Edition as they are reading or following along in class.

About the Microsoft Technology Associate (MTA) Certification

Preparing Tomorrow's Technology Workforce

Technology plays a role in virtually every business around the world. Possessing the fundamental knowledge of how technology works and understanding its impact on today's academic and workplace environment is increasingly important—particularly for students interested in exploring professions involving technology. That's why Microsoft created the Microsoft Technology Associate (MTA) certification—a new entry-level credential that validates fundamental technology knowledge among students seeking to build a career in technology.

The Microsoft Technology Associate (MTA) certification is the ideal and preferred path to Microsoft's world-renowned technology certification programs. MTA is positioned to become the premier credential for individuals seeking to explore and pursue a career in technology, or augment related pursuits such as business or any other field where technology is pervasive.

MTA Candidate Profile

The MTA certification program is designed specifically for secondary and post-secondary students interested in exploring academic and career options in a technology field. It offers

students a certification in basic IT and development. As the new recommended entry point for Microsoft technology certifications, MTA is designed especially for students new to IT and software development. It is available exclusively in educational settings and easily integrates into the curricula of existing computer classes.

MTA Empowers Educators and Motivates Students

MTA provides a new standard for measuring and validating fundamental technology knowledge right in the classroom while keeping your budget and teaching resources intact. MTA helps institutions stand out as innovative providers of high-demand industry credentials and is easily deployed with a simple, convenient, and affordable suite of entry-level technology certification exams. MTA enables students to explore career paths in technology without requiring a big investment of time and resources, while providing a career foundation and the confidence to succeed in advanced studies and future vocational endeavors.

In addition to giving students an entry-level Microsoft certification, MTA is designed to be a stepping stone to other, more advanced Microsoft technology certifications.

Delivering MTA Exams: The MTA Campus License

Implementing a new certification program in your classroom has never been so easy with the MTA Campus License. Through the purchase of an annual MTA Campus License, there's no more need for ad hoc budget requests and recurrent purchases of exam vouchers. Now you can budget for one low cost for the entire year, and then administer MTA exams to your students and other faculty across your entire campus where and when you want.

The MTA Campus License provides a convenient and affordable suite of entry-level technology certifications designed to empower educators and motivate students as they build a foundation for their careers.

The MTA Campus License is administered by Certiport, Microsoft's exclusive MTA exam provider.

To learn more about becoming a Microsoft Technology Associate and exam availability, visit www.microsoft.com/learning/mta.

Activate Your FREE MTA Practice Test!

Your purchase of this book entitles you to a free MTA practice test from GMetrix (a $30 value). Please go to www.gmetrix.com/mtatests and use the following validation code to redeem your free test: MTA98-374-AF44E1E15E17.

The **GMetrix Skills Management System** provides everything you need to practice for the Microsoft Technology Associate (MTA) Certification.

Overview of Test features:

- Practice tests map to the Microsoft Technology Associate (MTA) exam objectives
- GMetrix MTA practice tests simulate the actual MTA testing environment
- 50+ questions per test covering all objectives
- Progress at own pace, save test to resume later, return to skipped questions
- Detailed, printable score report highlighting areas requiring further review

To get the most from your MTA preparation, take advantage of your free GMetrix MTA Practice Test today!

For technical support issues on installation or code activation, please email support@gmetrix.com.

Acknowledgments

MOAC MTA Technology Fundamentals Reviewers

We'd like to thank the many reviewers who pored over the manuscript and provided invaluable feedback in the service of quality instructional materials:

Yuke Wang, University of Texas at Dallas
Palaniappan Vairavan, Bellevue College
Harold "Buz" Lamson, ITT Technical Institute
Colin Archibald, Valencia Community College
Catherine Bradfield, DeVry University Online
Robert Nelson, Blinn College
Kalpana Viswanathan, Bellevue College
Bob Becker, Vatterott College
Carol Torkko, Bellevue College
Bharat Kandel, Missouri Tech
Linda Cohen, Forsyth Technical Community College
Candice Lambert, Metro Technology Centers
Susan Mahon, Collin College
Mark Aruda, Hillsborough Community College
Claude Russo, Brevard Community College
Heith Hennel, Valencia College
Adrian Genesir, Western Governors University
Zeshan Sattar, Zenos
Douglas Tabbutt, Blackhawk Technical College
David Koppy, Baker College
Sharon Moran, Hillsborough Community College
Keith Hoell, Briarcliffe College and Queens College—CUNY
Mark Hufnagel, Lee County School District
Rachelle Hall, Glendale Community College
Scott Elliott, Christie Digital Systems, Inc.
Gralan Gilliam, Kaplan
Steve Strom, Butler Community College
John Crowley, Bucks County Community College
Margaret Leary, Northern Virginia Community College
Sue Miner, Lehigh Carbon Community College
Gary Rollinson, Cabrillo College
Al Kelly, University of Advancing Technology
Katherine James, Seneca College
David Kidd, Western Governors University
Bob Treichel, Lake Havasu Unified School District & Mohave Community College

Brief Contents

www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

Contents

www.itbookshub.com

www.wiley.com/college/microsoft *or*
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

LESSON 1

Ideating and Conceptualizing a Game

EXAM OBJECTIVE MATRIX

SKILLS/CONCEPTS	MTA EXAM OBJECTIVE	MTA EXAM OBJECTIVE NUMBER
Ideating a Game	Differentiate among game types. Differentiate among game genres. Understand the different game platforms.	1.1 1.2 2.5
Creating the Game Concept	Understand player motivation.	1.3

KEY TERMS

arcade
controllers
game concept
game mechanics
gameplay
game setting
game type
genre
handheld game console
massively multiplayer online role-playing game (MMORPG)
mission statement
mobile device
motivation
multiplayer games
offline games
online games
single player games
storyline
target audience
user interface (UI)

FunGaming Inc., is a game development start-up company. The goal of the company is to research the gaming needs of individuals across the world and make gaming products that exhibit an innovative design, addictive gameplay, stunning graphics, and leading-edge technology. The organization is currently in the process of ideating and conceptualizing one such product. During this phase, the project team identifies the game's target audience and decides the game type and genre accordingly. The team then creates the game concept, which includes the mission statement and storyline of the game, gameplay, and game mechanics.

■ Ideating a Game

THE BOTTOM LINE

Identifying the idea of a game is the first step in the game development process. The idea for a game encapsulates defining the ***motivation*** for the set of players you are targeting and deciding the genre and platform that is most appropriate for those players.

Games are one of the means of social interaction across all human cultures. Digital games are a new face of this age-old method of social interaction. Developing an interesting game is a challenging task. It might appear daunting at first, especially if you want to create a game similar to the ones available off-the-shelf, complete with complex animations and elaborate programming. However, if you follow a set procedure and consider a few elements of the game concept, the complexity of the process reduces considerably.

A good game begins with a strong idea, which comes from understanding what motivates people for playing games.

Identifying the Motivation

Identifying the motivation behind playing games involves finding the reasons why people play video games.

Playing video games has become a universal trend. People play and get addicted to the games that satisfy some of their basic needs. As a game developer, you should first identify those basic needs. The next step is to identify the selective needs of the players that you want to fulfill through your game.

Playing games satisfies three psychological needs of people: achievement, recognition, and satisfaction. People enjoy the challenges of a mentally stimulating game and are extremely delighted when they achieve the goals of the game. By winning the game, they earn recognition and respect from others. This in turn gives them great satisfaction.

Games also satisfy one of the most important emotional needs of people: the need to connect and interact with others. Games enable social interaction across geographical and cultural boundaries.

These psychological and emotional needs manifest into one or more of the following basic needs:

- Quest
- Learning
- Task management
- Determination
- Competence
- Thrill

QUEST

One benefit that encourages people to play video games is that they can gain new knowledge and skills. For example, novice cricket or basketball players might want to play a video game to experiment with new strategies on the screen before trying them out on the field. This helps them to improve their performance when they implement the strategy on the field and mentally prepares them to face their opponents with clear resolutions.

LEARNING

Good video games provide an excellent learning environment by providing personal control and autonomy to players. Players can explore new topics and learn to deal with the challenges imposed by video games by experimenting with their actions without worrying about the consequences. The losses sustained in making a mistake in a game world are much less significant than those sustained in the real world. This in turn provides self-confidence, which is a key ingredient of the learning process.

TASK MANAGEMENT

Video games involve resource and attention management. Players plan and track goals and subgoals, all in pursuit of winning the game. Identifying and managing the game tasks successfully provide the players with a sense of achievement.

CHALLENGE

Challenges in the game play a key part in motivating the players. However tough the challenge, hard-core players might not be willing to give up until they achieve the game's goal. Progressing through the different game levels can be annoying. With constant perseverance, players achieve the predefined game goal and feel proud of their achievement.

COMPETITION

Video games provide an outlet to the players to compete by overcoming interesting challenges. This not only fulfills the basic need for competition but also helps to determine and proclaim how competent the winner is in handling a given challenge. Additionally, the game encourages players to compare their gaming achievements with their peers and decide the best player among them.

THRILL

Video games can simulate situations that players might never experience in real life. In a virtual world, everyone can be a hero and save a victim from the villain by shooting the villain in the head. Or a player may be able to act as a detective and solve the most difficult case better than Sherlock Holmes himself. Or an earthquake victim can experience the thrill of finding a way out of a devastated city. Or a Formula 1 driver can win a close race. These experiences thrill the players and satisfy their need for exploring the unimaginable.

Identifying the Target Audience

The game motivation varies with the type of players. While ideating for a game, it is important to identify the ***target audience*** or group of people for whom the game is meant.

To develop an effective game for a given audience, it is essential to become familiar with the likes and dislikes of your target audience. The target audience for games can be distinguished based on age and gender. Based on age, players are categorized into preschoolers, children, teens, and adults. Based on gender, players can be male or female. Players can also be divided into groups based on demographic factors, such as geographical location, nationality, culture, and religion. Each of these player groups have different needs and motivational levels to play games.

Based on the needs and motivational levels, the audience for games can be further divided into casual players, hard-core players, intermediary players, and professional players. Let us look at this variety of players in more detail.

CASUAL PLAYERS

Casual players are people who:

- Have limited interest in playing games
- Do not spend much time playing games
- Prefer to purchase easy games and buy difficult games rarely, only if they are influenced

Casual players are older and mostly female. However, given that this group likes easy games, developing games for casual players might not require much use of your creativity and intellect. Casual players play games such as *Jawbreaker, Minesweeper,* and *Solitaire*.

HARD-CORE PLAYERS

Hard-core players are people who pursue games seriously. They are people who:

- Have a passion for games
- Like to play high-level or challenging games
- Tend to play more involved games that require more time to complete
- Tend to compete in tournaments and leagues

Hard-core players do not attempt to purchase easy games because they are rigid due to their high expectations. With their high demand, they prompt your creativity and progression by kindling new ideas and creative solutions. Hard-core gamers play games such as *Halo: Combat Evolved Anniversary, Forza Horizon,* and *Metal Gear Solid 4.*

INTERMEDIARY PLAYERS

Both casual players and hard-core players are two extreme customers in the gaming industry. In contrast, intermediary players are in the middle, between the casual and the hard-core players. The intermediary players are people who:

- Have an interest in diverse games
- Spend less time playing games compared to hard-core players
- Like to play games that are neither too easy nor too difficult
- Do not intend to take part in competitions

Players in the age group of 16 to 22 years around the world fit well into this category. Developing games for intermediary players involves creating balanced games that are not too simple or too complex. This enables maintaining the balance in terms of both creativity and profitability. Intermediary players play games such as *Quantum Redshift, Kinect Adventures*, and *Kinect Sports*.

PROFESSIONAL PLAYERS

Professional players are hard-core players who play games to earn money. Many companies pay these players for testing the games developed by them. Game testing is now an important line of business in the games industry.

Identifying the Game Genre

Video games are classified into various genres based on the type of player's interaction with the game called ***gameplay***.

CERTIFICATION READY
Which game genre do you suggest if your target audience consists of children in the age group of 10 to 14 years?
1.2

The word ***genre*** refers to a particular class or type of an artistic venture. For instance, cinema has different genres, such as action, comedy, and romance. Similarly, you can also categorize games into different genres.

You decide the genre of the game based on the motivation and the target audience identified for the game. For example, young children play games for fun, which enables them to learn concepts such as counting and spelling. These games belong to the educational games genre. An example of an educational game is *The Magic School Bus in the Time of the Dinosaurs.* Besides the educational games genre, other genres include the following:

- Action
- Adventure
- Sports
- Simulation
- Role-playing game (RPG)
- Fantasy
- Card
- Board

ACTION

Action games, as the name suggests, focus on warfare. In this game genre, players make rapid moves to carry out attacks on targets. Players need spontaneous and quick reactions to overcome game challenges. Two subtypes of this genre are the shooter game and the fighting game.

A shooter game involves destroying all opponents using some sort of weapon, usually a gun or some other shooting weapon. The aim of this game is that the shooter or the player character remains alive throughout the mission. There are two viewpoints through which the player can view the events in this game. Based on these viewpoints, the shooter game genre further divides into the following:

- **First-person shooter (FPS) genre:** In FPS, the player can view the events through the shooter character's eye. *Halo 4: Forward Unto Dawn* is an example of the FPS subgenre (see Figure 1-1). Though a stealth action game, it can be played at many places with a first-person camera view.
- **Third-person shooter (TPS) genre:** In TPS, the player views the events through a camera that follows the shooter character from a few feet behind. *Gunstringer* is an example of the TPS subgenre (see Figure 1-2).

A fighting game involves knocking the opponent unconscious or dead through different moves. *Street Fighter* is an example of a fighting game.

Figure 1-1

Halo 4: Forward Unto Dawn, an example of a first-person shooter game

©Microsoft Corporation

Figure 1-2

Gunstringer is a third-person shooter game

©Microsoft Corporation

ADVENTURE

An adventure game does not involve spontaneous challenges or action. Rather, the player solves different puzzles by interacting with the characters or the environment of the game. Some examples of adventure games are *Myst, Nick Tethers: Puzzle Agent*, the *Harry Potter* game series, the *Fable* game series*, Kinect Adventures*, and the *Maw.* Figure 1-3 and Figure 1-4 show screens from *Kinect Adventures* and *Fable II,* respectively.

SPORTS

Sports-based games simulate traditional sports, such as baseball and snooker. Some sports-based games focus on the actual playing of a sport, whereas certain games, such as the *Championship Manager*, stress the strategy behind the sport. In addition, games such as *Arch Rivals* mock sports for comedic effect. A few of the sports games characterize real sports champions to excite the players and update periodically to reflect real-world changes. An example of a sports game is *Kinect Sports: Season 2* (see Figure 1-5).

Figure 1-3

The *Kinect Adventures* game

©Microsoft Corporation

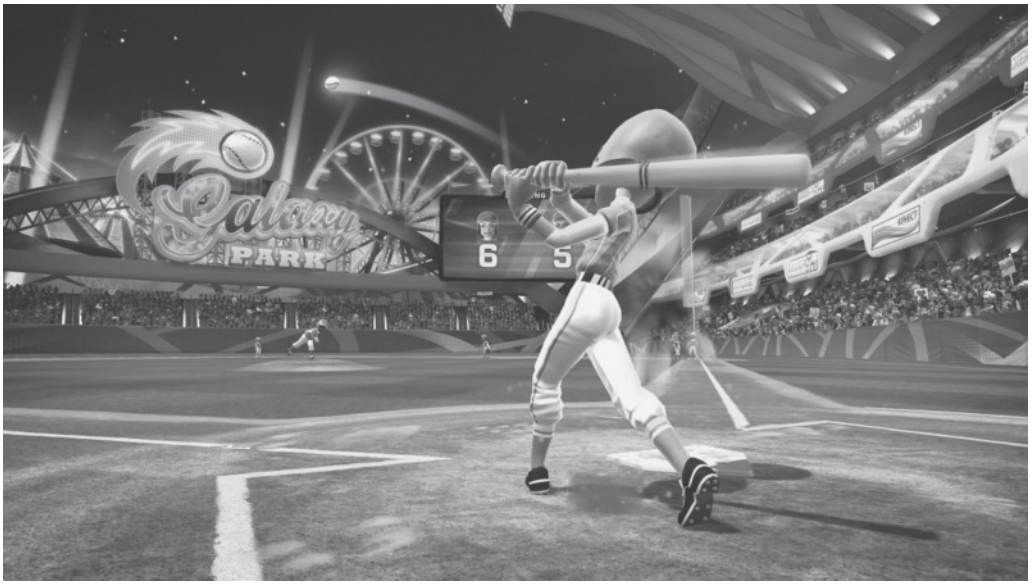
Figure 1-4

The *Fable II* game

©Microsoft Corporation

Figure 1-5

The *Kinect Sports: Season 2* game



©Microsoft Corporation

SIMULATION

A simulation game emulates real or fictional activities. This genre provides the player with a realistic analysis of the object involved in the game. For example, in a flight or vehicle simulation game, the player gets to fly the aircraft or the vehicle as realistically as possible. Some of the games in this genre include *Sim City* and *Combat Flight Simulator 3* (see Figure 1-6).

ROLE-PLAYING GAME (RPG)

This game type makes the player play the role of one or more characters in a predetermined storyline. Each character has specific skills and abilities. The player explores and completes the quests to access unique locations such as dungeons and castles, and amass possessions such as gold or weaponry. Examples in this genre include *Lost Odyssey* and *Fable Heroes* (see Figure 1-7).

Figure 1-6

The *Combat Flight Simulator 3* game

©Microsoft Corporation

Figure 1-7

The *Fable Heroes* game

©Microsoft Corporation

FANTASY

Fantasy games are based on an imaginary virtual world. Examples include *Final Fantasy* and the *Baldur's Gate* series.

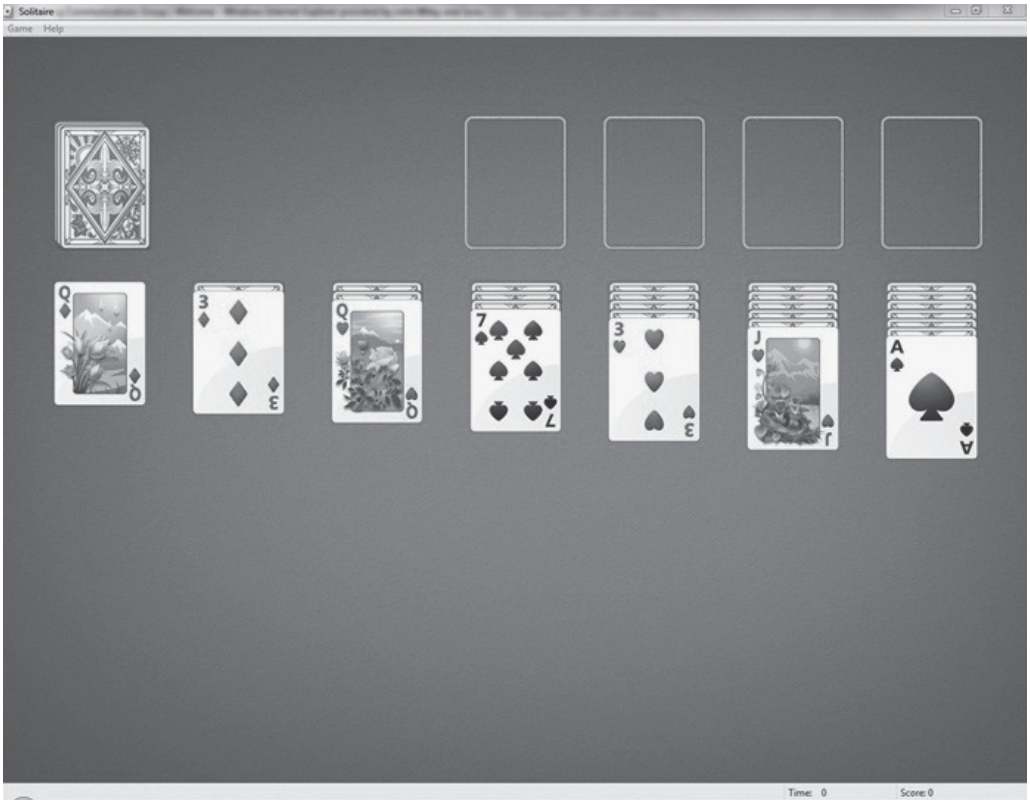
CARD

This game genre uses virtual playing cards as the main device. Certain card games use traditional decks and apply standardized rules, whereas others are folk games in which the rules vary by location and culture. A popular card game that uses standardized rules is *Solitaire* (see Figure 1-8). *French Tarot*, a card game that is popular in France, uses rules published by Fédération Française de Tarot.

www.itbookshub.com

Figure 1-8

The *Solitaire* game



BOARD

This game involves counters or objects moved or placed by a certain set of rules on a virtual playing board. It requires players to use strategy, chance, or a combination of both, until they achieve their goal. *Backgammon* is an example of a video board game (see Figure 1-9).

Figure 1-9

The *Backgammon* game

Identifying the Game Type

CERTIFICATION READY
What are the different types of games?
1.1

Games can be of various types: online or offline; single player or multiplayer; and console, PC, arcade, or mobile games.

The term ***game type*** is used in the gaming industry to mean different things in different contexts. Some people use the term to mean game genre, whereas others use it to mean the gaming platform or the electronic device that, in conjunction with software, runs digital games. Still others use it to mean the status of the game, online or offline, or the nature of the game, single player or multiplayer. Without challenging any of these definitions of game types, let us look at the different types of games.

GAME STATUS

Based on the status, games can be of two types:

- ***Online games*** are video games available on the Internet. Players can log on to the Internet and play these games for free or at a certain cost. Online games include *Zone of War.*
- ***Offline games*** are video games that come in the form of disks (such as a CD or DVD), in memory cards, or as downloads from the Internet. Players can play offline games either on a PC, on a video game console, or on their mobiles. An example of an offline game is *Final Fantasy.*

NUMBER OF PLAYERS

Games can be divided into two types based on the number of players: single player games and multiplayer games.

As the name suggests, ***single player games*** involve a single player. The player plays with the preprogrammed challenges to reach the goal. The major selling points of single player games are interesting storylines, impressive graphics, and realistic nonplayer characters and opponents. Notable examples include action-adventure games such as *The Legend of Zelda*, platform games such as *Mario and Sonic*, and stealth games such as the *Metal Gear Solid* series.

Multiplayer games involve more than one player. Multiple people can play in the same game environment simultaneously and interact with their peers by cooperating or competing with each other in order to reach a common goal. Multiplayer games are becoming increasingly popular with gamers. These games provide the gamers with a form of social communication that is usually missing in single player games. In a variety of different multiplayer game types, players may individually compete against multiple opponents, work cooperatively with one or more partners in order to achieve a common goal, supervise activities of other players, or engage in a game type that incorporates any possible combination of these varieties. An example of a multiplayer game is *Halo: Combat Evolved Anniversary.*

One of the popular multiplayer game types includes ***massively multiplayer online role-playing game (MMORPG).*** Similar to any other RPG game, MMORPG also enables players to assume the role of a character and control many of its actions. However, unlike most RPG games, MMORPG involves hundreds of players interacting with each other and sharing the same game resources in a persistent virtual world. This virtual world persists even after the gamer has logged off. An example of an MMORPG is *The Lord of the Rings Online: Shadows of Angmar.*

An MMORPG uses client-server architecture. A server runs the instance of the game's virtual world that is usually hosted by the publisher of the game. Players connect to this virtual world using client software saved on a disk or through a Web browser. For example, *World of Warcraft* uses client software saved on a disk, whereas *Ragnarok* accesses the virtual world through a Web browser specifically designed for that game.

Certain MMORPGs use multiple servers to run independent virtual worlds. A player using the virtual world running in one server cannot access another virtual world running on another server.

GAMING PLATFORM

CERTIFICATION READY
Which game platform is ideal for a person who travels a lot and wants entertainment during transit?
2.5

In the early days, games were stored in cartridges. The players loaded the cartridges on interactive electronic devices to run these games. They used the television as the display device. With the advancement of technology and advent of the Internet, new gaming platforms have emerged and the existing ones have become more sophisticated. Today games can be played on varieties of platforms, online or offline, by single or multiple players simultaneously. Some of the common game platforms that you can select from are:

- Console
- PC
- Arcade
- Mobile

CONSOLE

CERTIFICATION READY
What are the most common devices used with game consoles?
2.5

A console is an interactive entertainment machine, primarily used to run video games. This device usually connects to a television to display the visual output of the game. The player usually operates and controls the game using a controller. ***Controllers*** are handheld devices, such as a remote control connected to the console. The player interacts with the images on the screen by operating the buttons and directional controls contained in the controllers.

Console games come in the form of CDs and DVDs that can be inserted directly into the console device. Nowadays, players can also download games directly to the console through the Internet. However, to play a specific console game, players need to have the respective console device for which the game was designed. One commonly used console device is Xbox 360 (see Figure 1-10). Xbox 360 is Microsoft's video game console. It has built-in hard disk drives to store games and content downloaded from Xbox Live, an online gaming service from Microsoft. Other console devices include PlayStation 3 produced by Sony Computer Entertainer and the Wii GameCube produced by the Japanese company Nintendo Co., Ltd.

Figure 1-10
An Xbox 360 game system

©Microsoft Corporation

A ***handheld game console*** is a lightweight portable device, which the players can carry anywhere and use to play a console game anytime. In this device, all the units such as the display unit, the controller, and the audio unit are built in. Popular handheld game devices include Nintendo 3DS from Nintendo and PlayStation Vita and PlayStation Portable (PSP) from Sony.

Multiplayer console games can be played over the Internet as previously described, or multiple controllers can be connected to the video game console. The output display is divided among the multiple local players.

PC

CERTIFICATION READY
Which game types are best suited for the PC platform?
2.5

A game running on a PC is called a *computer game* or a *PC game*. PC games are published in various forms, such as DVDs, CDs, or downloadable content. Depending on the game, PC games sometimes require special hardware in the player's computer or an Internet connection to play the game online. PC games have increased playability as both newer and older games are played on any version of the software system. PC games can be single player or multiplayer.

ARCADE

CERTIFICATION READY
What are arcade games?
2.5

An ***arcade*** is a coin-operated entertainment machine (see Figure 1-11) found in public places such as theaters and restaurants, and especially in amusement parks. *Dance Dance Revolution* and *DrumMania* are examples of some popular arcade games.

Arcade games often have more control accessories than PC or console games. In addition, as arcade games are played in public places and provide a form of social hangout, these games emphasize individual performance over the content of the game.

Arcade games can be both single player and multiplayer games. In multiplayer arcade games, each player can have his or her own game controls connected to the arcade.

Worldwide, the revenue from the arcade game industry showed steady growth until 2002. In recent years however, and particularly in Western countries, the use of arcade games has

Figure 1-11
An arcade game machine

©jamesbenet/iStock Photography

considerably declined. Arcade gaming remained a flourishing industry in China and Japan until the recent economic recession caused a marked decline in Japanese arcade gaming.

MOBILE

CERTIFICATION READY
What are the types of mobile devices?
2.5

A ***mobile device*** is a handheld device that includes cell phones, smartphones, tablet computers, and PDAs. Players can load mobile games in their cell phones via Bluetooth from the mobile operator's network or via memory cards at the time of purchase. Additionally, they can also download games by accessing the Internet. The mobile games run on the existing platform available on the mobile phones.

CERTIFICATION READY
Which game types are suitable for the different game devices?
1.1

Although many mobile games are single player games, there are also multiplayer mobile games. In a multiplayer mobile game, players can connect through various communication channels, such as Infrared, Bluetooth, Wi-Fi, Wireless LAN, or GPRS (general packet radio service; commonly referred to as 2G, 3G or CDMA wireless bands). Mobile phones with GPRS connection share data globally. You can connect a large number of mobile games with a single server that acts as a router. Players can share data by connecting to the server. Alternatively, cross-platform games use GPRS, in which the game allows the mobile gamer to play against a PC gamer.

■ Creating the Game Concept

THE BOTTOM LINE

Creating a ***game concept*** or conceptualizing your game means giving a concrete shape to your ideas for the game. This process involves creating a mission statement, storyline, gameplay, and mechanics of the game.

CERTIFICATION READY
What is the significance of conceptualizing a game?
1.3

Generating an idea for a game is just the beginning of the creative process. Listing your ideas, fleshing them out, setting the rules of the game, finalizing the characters, and many other activities go into the making of the final product.

A game concept is the elaboration of the game idea. It includes jotting down the following:

- The final objective of the game or the mission statement
- The plot or the storyline
- Various interactive and noninteractive elements of the game or the gameplay
- The flow of the game or game mechanics

To know how conceptualization works, assume that you have to create the character of a joker, which will be one of the interactive elements in the game. Before you can design this character, you need to conceptualize it. A joker is usually associated with characteristics such as funny, colorful, sociable, outgoing, and other traits associated with goodness. However, if the idea is of an evil joker, then you can conceptualize the same character with characteristics and looks that are not associated with the usual goodness of a joker. Eventually, you will create a character similar to the evil joker from *Batman*.

All elements of the game are conceptualized in the same manner. In the following sections, we learn how to create the game concept through simple steps. Figure 1-12 displays the game concept creation process that shows the various steps and elements.

Writing the Mission Statement

A ***mission statement*** describes the main objective of the game, challenges involved in the achievement of that objective, and the resources available for the players to help achieve that objective.

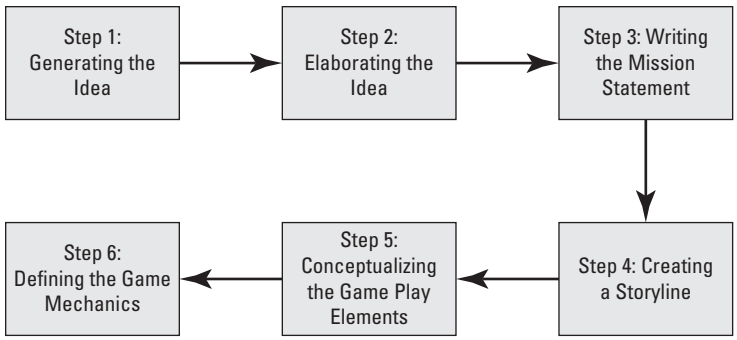


Figure 1-12

The game concept creation process

The purpose behind writing a mission statement is to clearly pass on the idea of the game to the game production team. While writing the mission statement, you must focus on the formal elements of your game, which are the underlying system and mechanics of the game. A basic mission statement must answer three questions:

- What is the objective of the game?
- What challenges does the game present to the player?
- How will the player achieve the objective?

To write a good mission statement, you need to ask yourself the following questions:

- How should the player act and feel in the game?
- What role should the player perform in the game?
- Does the player have a well-defined objective?
- What are the obstacles in achieving the objective?
- What kind of resources will the player have?

The previous list of questions is not exhaustive. You need to analyze your idea and arrive at answers that will eventually lead to a refined mission statement.

A good mission statement is the benchmark to test all the ideas that you consider in the game. The final version of the game must not have any element that goes against the mission statement.

You can rely on the following tips to write the mission statement for your game:

- Involve everyone connected to the game and brainstorm together. Others also can provide useful insights.
- Set aside several hours to work on your statement.
- Make every word of the mission statement count.
- Avoid making a generic mission statement.

Let us look at an example of a good mission statement. Suppose you are developing an adventure game and you decide to create the game set of a forest. The mission statement for your game can be:

"Creating a game that will be set with challenges not just from your opponent, but also from nature. The only way to win is to survive, but remember that the tyrant is watching."

As per the mission statement, the objective is to survive against all odds. The challenges involved in achieving that objective are an opponent player and elements of nature designed for the game. The mission statement also presents the method to achieve the objective, which is to live or ensure that the player's character remains alive at the end of the game. This mission statement describes the intended gamer experience and states what the gamer should feel or think.

Creating a Storyline

The ***storyline*** of a game is the underlying plot that defines the flow of the game from start to finish. A good storyline is the primary ingredient for creating an engaging game.

Just like movies, games also tell a story. These stories revolve around human interests: love, family, threats, adventure, suspense, and death. Similar to all stories, the game narrative or storyline is an expression of human imagination.

Players often prefer a compelling and interesting story. If you want a great engaging game, you must have a good storyline behind it.

Here are some examples of a game storyline:

- **Counter Strike:** Different Counter Terrorist squads fight to rescue the hostage alive from the clutches of different terrorist organizations.
- **Super Mario Bros:** A player controls Mario or Luigi, a character in the game. The player makes him walk, run, or jump, while avoiding traps, overcoming obstacles, and collecting treasures in order to save the princess from the dragon.
- **Diablo:** Players battle monsters, seek treasures, and explore dungeons in an attempt to amass wealth and become powerful.

Writing a storyline is difficult. You must write a story that is well-paced and has defined objectives. Through your story, you must create a world that is as believable as the real world that we live in. A believable world helps the player forget that he or she is playing a game and boosts the gaming experience.

PARTS OF A STORYLINE

A storyline is easier to write when you break it into two parts: purpose and complexity.

The first thing that you must focus on when writing the storyline is the purpose of the story in the game. Do you want the storyline only to introduce the plot or also take the player through the different levels of the game? To engage the player through your storyline, the purpose has to map to the motivational needs of the players. Only when the storyline maps to the deepest needs of the players will the players be able to feel connected with the game.

The second thing to consider is the complexity of the storyline. A storyline must be neither too complex nor too simple. Designers often ignore the storyline while focusing on graphics. However, players take the storylines seriously. Therefore, you must strike a balance between the narrative and the graphics to make the storyline attractive.

One way to add complexity in the storyline and make it interesting is to use a twist in the story. The twist keeps the player engaged and eager to continue playing to uncover what happens next. For example, in *Hitman*, the main character Hitman has something in his past that drives him to become a Hitman. Another twist is that during the course of the game, Hitman himself becomes a target and he needs to find out the person in his organization who has set him up as the target.

TAKE NOTE*

You must exercise caution when using the twist strategy. The twist must make logical sense. It must not confuse, frustrate, or disengage the player from the game.

Twists often make a storyline complex and must, therefore, be handled with care.

Sometimes, in their zeal to make the storyline simple, designers often play it safe and fall back on time-tested, repeated storylines. Such a storyline might become uninteresting for the players. You must always create storylines that mirror the complexity of the game. For example, simple puzzle or arcade games such as *Tetris* do not have any storyline, whereas *Aladdin* has a narrative throughout the game.

COMMON MISTAKES

Although it is easy to decide on the purpose of the storyline, designers often tend to make mistakes on the complexity. Either they make it too complex or too simple. A storyline that is too complex can irritate the player as much as a simple storyline might bore the player. Therefore, to write a good storyline, you must avoid making the following common mistakes:

- **Too much dependence on lovable, friendly characters:** Although it might seem a good way to add depth and drama to your game, you must use such characters carefully. Video games that have virtual pets for players to take care of as part of the games are common in China and Japan. Also, games such as *Final Fantasy* that have lovable and friendly characters are quite popular.

  However, there might be players who feel that in reality, we do things with or for our friends because we enjoy their company. These players might get tired of playing games with a robot. A careful analysis of your target audience will help you decide on the use of friendly characters.
- **Fetch quests:** When you design a twist in the story by making the player perform a task to move forward in the game, it is called a *fetch quest* ploy. For example, in a game, the central character reaches a locked door. To open it, the player needs to find the key. An old man is the keeper of the key who will give the player the key only if he or she successfully completes the task that the key keeper sets.

  This ploy of fetch quests is often attractive to designers who think that because the player is having fun, give the players more fun. However, it irritates the players, especially if introduced when the game is about to end. Making the player go in circles just to make the game longer brings the story to a halt. It will eventually lead to frustration for the player.
- **Turnarounds:** Turnarounds make the players return the same way they might have come so far—that is, to trace the path back to the starting point. And on the way back, players have to solve the same puzzles and avoid the same enemies. For some games, each level of the game starts from the same point. Designers might want to use this to make the player appreciate the high-level graphics and the effort spent to create that stage. But players, who were probably enjoying the game until now, might feel completely lost and frustrated.

  Turnarounds work well in the case of adventure puzzle games. For example, in *Tomb Raider*, which is an action-adventure series, the game character Lara has to find a particular object to unlock a specific door. Once she finds the object, she has to trace the path back to the door that has to be unlocked. However, turnarounds become annoying for most of the other game types unless executed with a proper gameplay.

Conceptualizing the Gameplay

A ***gameplay*** describes various elements through which the player interacts with the game. These include the visual theme and cinematic, objects and characters in the game, user interface, and the audio theme.

Gameplay consists of the visual elements that help players to maintain their belief in the game, its characters, and objects, resulting in a more immersive experience. Gameplay is largely influenced by two things: game setting and the storyline of the game.

Game setting is the term commonly used to refer to the procedures that players use for setting up or personalizing a game. Examples include typing in the player's personal information, such as the player's name, and choosing the character that the player wants to play. Some games even allow players to design their avatars. For example, *Halo* allows feature customization for characters. Most games today allow customizing user interface elements. For such customizable games, the gameplay element largely changes with the game setting.

Just as the game setting influences the gameplay elements, the storyline of the game also has an impact on the gameplay elements. The storyline is the birthplace of gameplay because the storyline sets the mood of the game. A strong storyline creates an excellent game experience with various gameplay elements in the game.

There are two types of gameplay, linear gameplay and nonlinear gameplay. In a linear game, the challenges are predetermined in a fixed sequence. The gamers have to follow the predetermined plot in this gameplay type. *Halo 3* and *Call of Duty 4* are examples of the linear gameplay.

In a nonlinear game, the challenges are posed to the player in multiple sequences. The gamers can choose their choice of path to victory. *Borderlands* and *Alpha Protocol* are examples of non-linear gameplay.

While conceptualizing a game, you conceptualize the following gameplay elements:

- Visual theme and cinematic
- Objects
- Characters
- User interface (UI)
- Audio theme

Let us look at these elements in more detail.

VISUAL THEME AND CINEMATIC

The visual theme helps set the stage for the game. Creating a visual theme involves deciding a common background, color pallet, and effect for all the visual elements of the game, so that they all look like they belong to the same game.

The visual theme has to go hand-in-hand with the idea of the game. This means that for a reality-based game, you need to thoroughly research the background to create a visual theme. For example, for a game based on the history of the freedom struggle of India, you need to know the correct historical details of that era. In contrast, for fantasy-based games that are set in an imaginary world, you can depend on your creativity to flesh out the details of the imaginary world. For example, in the *Final Fantasy* game, an entire new world was conceptualized.

A part of the visual theme is cinematic in nature. Cinematic or cut-scenes refer to the sections in the game where the player does not have any control—that is, the non-interactive sequences of the game. Such sequences are aligned with the visual theme and help make the game story progress, introduce characters, or provide information to the players. According to the purpose and the theme of the game, cut-scenes can be entertaining, informative, dramatic, or thrilling. For example, *Pac-Man*, a hit game from the year 1980, was the first game to use cut-scenes in the form of animated interludes between certain stages of the game.

Before creating a cinematic sequence, you must outline the goals by answering the following questions:

- Why must the player watch the cinematic elements?
- What kind of reaction from the player is the cinematic intended to elicit?
- Will the cinematic inform the player about the game and the characters?
- Do you want the cinematic to showcase the game's graphics?

OBJECTS

Objects are the visual elements of a game that complete a scene in the game and make it look real. For example, in a car racing game, all the elements that constitute a gas pump where the car stops to take fuel are objects. The player may or may not be able to interact with all the objects. For example, the player may be able to click the hose to fill up the car but may not be able to swipe the credit card machine placed in the scene.

Figure 1-13
An example of an object

©26ISO/iStock Photography

At the conceptualization stage, the designer identifies the objects for each scene in the game and passes it on to the concept artist, who will design these objects keeping the visual theme in mind. Figure 1-13 shows an example of an object.

CHARACTERS

In most of the games today, a character represents the player. Therefore, it is important to conceptualize a character in the most realistic manner so that the player can recognize himself or herself through that character.

Conceptualizing a character involves fleshing out your character's persona. This broadly includes conceptualizing the character's looks, image, actions, and dialogs.

Finding answers to the following questions will help you create your character's persona:

- How will the physique of the character be?
- What does the character eat?
- Where does the character live?
- What is the character's routine?
- How does the character behave when angry or surprised?
- Does the character have a unique catchphrase?
- How does the character feel about the game situation?
- Who is the character: villain, hero, mentor, ally, or someone else?

A lot of effort goes into conceptualizing every action a character needs to perform in a game—such as walking, running, jumping, and climbing—in order to enable the character to perform various tasks. The interplay and cohesion between the character's actions and dialogs result in bringing about the intended reaction from the player. The character's actions and words can transform him or her into someone whom the player likes or dislikes.

It is also important that, if need be, the character evolves in a believable manner. For example, Hulk transforms into a superhero because of gamma radiation, Spiderman because of a spider bite, and Iron Man because of his wealth. Figure 1-14 shows an example of a game character, from *Kinect Sports: Season 2*.

www.itbookshub.com

Figure 1-14

An example of a character

©Microsoft Corporation

USER INTERFACE (UI)

User interface (UI) is a collective term referring to the onscreen elements through which a player interacts with the game. The UI helps the player access information about the game world and the status of his or her character. The UI elements in a game include:

- Menus
- UI components (for example, different character avatars)
- Text
- Icons
- Layout
- Color

A well-designed UI makes the game easier to play. The player can easily assess the situation and respond accordingly. A poorly designed UI makes it harder for the player to identify what needs to be done or what resources are available, making it frustrating for the player and ruining a good game.

The UI of the game *Halo* is an example of a well-designed UI (see Figure 1-15). The game provides an interesting approach to the whole idea of UI.

Figure 1-15

The *Halo* game interface (UI)

©Microsoft Corporation

A well-defined UI design has the following characteristics:

- **Intuitive:** A good UI does not require the user to think about how to access or interpret the information and features of your game. Users should find what they need neatly and logically arranged for them on screen.
- **Responsive:** Players should receive appropriate feedback when they click or access UI elements so that they don't wonder whether the click or key press was successful.
- **Relevant:** You must be careful about how and when you use a particular type of UI element. For example, cluttering the UI with icons for different types of weapons might be useful in a game where the player has to frequently change the weapon to kill opponents. However, the same, cluttered UI will not suit a jigsaw puzzle game, in which the icons, although for useful purposes, might blend with the puzzle pieces and irritate the player. Similarly, color plays an important part in creating the correct gaming experience. Some colors are a distraction for the player and must be avoided on the UI.
- **User friendly:** The UI is not user friendly if it takes the player too many keystrokes or button presses to play the game. Players want to see important information at a glance, get threat alerts, and receive feedback for action taken through a UI that remains the same throughout the game.
- **Customizable:** Give the maximum possible control in the hands of players. For example, let them switch on or switch off the audio, no matter how important you perceive it to be a part of your game. You can give users the option of muting just the background music or just the sound effects. If you have a toolbox, let the players move it around and dock it where they find it most comfortable. Let them choose the window size. The list of things that you can let the player customize is endless; the essence is to let the players be happy and comfortable.

You must develop a prototype of the UI and carefully evaluate the individual elements of the UI for its intended effect on players.

AUDIO THEME

Can you imagine a scary movie or game with the sound turned down? It simply does not have the intended effect. For a game to have audio that engages the player, you must work with the sound designer. The sound designer advises on the audio elements that will work well with the proposed setting, story, and other gameplay elements.

Game audio also includes the sound that accompanies the non-interactive parts of the game (for example, the introduction movie and cut-scenes). It concerns elements of interactivity as well as audio outside the context of the game, such as sound for game trailers.

Game audio has evolved from singular beeps to soundtracks. To help you choose the best audio theme, you need to consider the following questions:

- Will each character have a unique voice?
- How does the characters' sound or dialog function in the game (for example, help for the player, comic relief, and so on)?
- What types of music work best with the game?
- Where in the game will the music play?
- What types of sound effects work best in the game?

The list is not exhaustive. You must remember that players enjoy the gaming experience if the game is supported by a good audio theme.

Remember the following tips when choosing the audio theme:

- Think about all the situations that can arise in the game and then consider the sounds you will require to go with the situations.
- Purchase and use sound software that will allow you to explore diverse sound effects.

- Consider the types of sound effects the game will require. For example, for a shooting game, you will need to create sounds that resemble gunshots, laser sounds, and general sounds.

Defining the Game Mechanics

Game mechanics is all about how a game operates. It includes the game rules, the challenges that a player will encounter as the game flows, the activities that the player will perform, and the goals that the player will achieve to win the game. Game mechanics also define how a player is to be declared a winner.

As a game designer, you need to realize that all games use mechanics; however, it is how the different elements of game mechanics interact with each other that makes a game complex or easy to play. For example, a simple FPS shooting game has simpler game mechanics than a complex action game.

To design effective game mechanics, you must first understand how the game works.

1. Essentially, in a game, a player performs an action.
2. The player's action triggers an effect in the game. This effect is based on the player's knowledge of the rules of the game. It is not possible for the player to know all the built-in rules of the game. The game designer has to conceptualize some portion of rules that will remain unknown to the player, subject to the player's eventual discovery. These hidden rules are referred to as the *black box*.
3. Based on the player's action, the simulation will provide a feedback for the player.
4. Depending on the feedback, the player will perform the next action.

Let us explore game mechanics with a simple example:

Assume that in a game, a player has to reach a target before the character's health deteriorates beyond a point. The player can improve the character's health by getting the character to eat apples on the way. To reach the apples, the character must be made to jump. In this game, the player will perform the action of making the character jump to eat the apple. This will in turn cause a game effect, which is determined by the success/failure and result of the player character's action. Based on the result of the player character's action and according to the game rule, the game will provide either of the following feedback.

- **Feedback1:** If the character reaches the apple, it will be shown as having been eaten, and this will trigger a corresponding improvement in the character's health.
- **Feedback2:** If the character cannot reach the apple, the apple will be shown to remain where it was on the screen and the character's health will remain the same.

Based on the feedback, the player will move the character forward or make the character jump again. Figure 1-16 shows the game mechanics cycle.

At the center of the game mechanics cycle are the various elements that you can use to engage the player. All games have the same mechanics; only their complexity differs. You must consider each of the following elements of game mechanics in your conceptualization:

- Quest
- Task
- Activities
- How to win
- Game goals

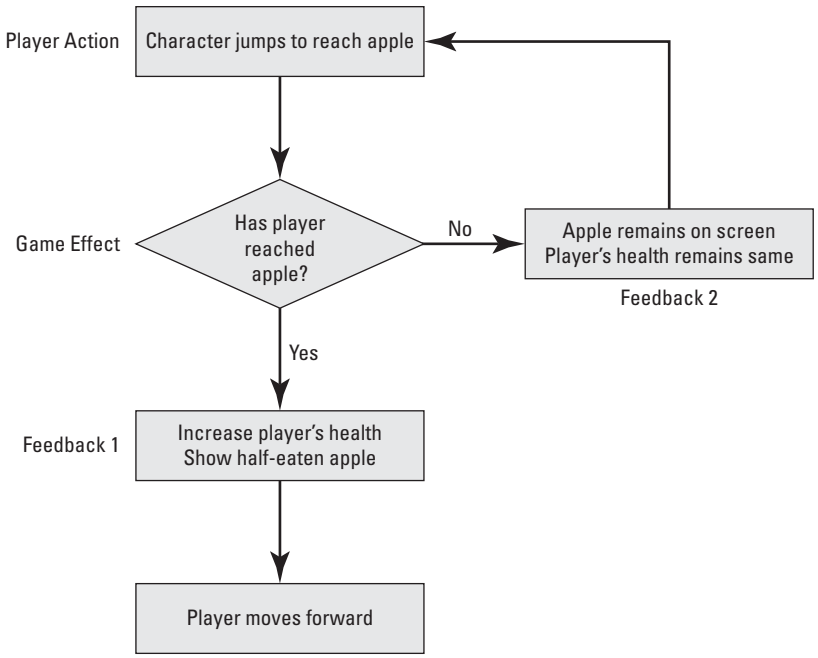


Figure 1-16
The game mechanics cycle

QUEST

CERTIFICATION READY
Can you name three different quests?
1.3

If you want the player to complete a set of predefined tasks in order to be successful in the game, you are essentially designing a quest mechanic. Quests can be used in adventure, fantasy, and role-playing games. In a quest, the player has to search a particular end result. A quest chain is a group of quests that are completed in sequence. Completion of each quest is a prerequisite to beginning the next quest in the chain. A quest usually increases in difficulty as the player progresses through the quest chain. An example of a quest is the set of tasks that the "Dragonborn" perform to achieve monetary rewards in the game *The Elder Scrolls V: Skyrim.*

Usually, quests lead to a reward mechanic in which the player is awarded some points for being successful. Rewards help in motivating the player to continue the game. It also enhances the playing experience.

There are different kinds of quests, such as:

- **Kill quest:** Specific characters need to be killed.
- **Collection quest:** The player has to collect certain objects.
- **Target quest:** The player has to reach a particular destination and deliver the goods.

You must remember not to overuse the quest mechanic. Nesting a quest within a quest may lead to the player getting irritated.

TASK

CERTIFICATION READY
What is a suitable task mechanic for an adventure game?
1.3

Task mechanics give the player a to-do list to complete a quest. Unlike quests, tasks might or might not end with rewards.

Here is an example of tasks from *Counter Strike*:

- **Task 1:** If you are a terrorist, kill all the opponents from the Counter Terrorist squad.
- **Task 2:** If you are a member of the Counter Terrorist squad, you need to kill the terrorists and save the hostages without letting anyone harm the hostages in any way.

You must refrain from loading your game with too many tasks because the player will eventually lose interest.

ACTIVITIES

Another game mechanic is activities. Activities are the subset of a task and suggest how the player will perform the particular task.

For example, if the task of a soldier is to kill a terrorist on the opposite side of the river, he will need to perform the following activities in the given order to complete that task:

1. The soldier rides on his horse.
2. The soldier crosses the river using an appropriate mechanism.
3. The solider strategizes and kills the terrorist.

HOW TO WIN

CERTIFICATION READY
What are three commonly used "how to win" mechanics?
1.3

Every player wants to win. The win happens when the player performs an action that results in a successful outcome. While conceptualizing a game, you need to define the conditions under which the player's action can result in a win. Some of the popular how-to-win mechanics are:

- **Loss avoidance:** For example, in chess, the winner of the game is the player who checkmates the other player's king, thereby securing his or her own king.
- **Eliminating opponents:** For example, in *Lord of the Rings: The Two Towers*, the player wins by killing all the monsters or enemies.
- **Logic:** For example, in Microsoft *Tinker*, the player wins when the puzzle is complete or the riddle is resolved.
- **Races:** For example, in *Snakes and Ladders* and *Ludo*, the first player to reach the finish line is the winner.
- **Scoring:** For example, in the case of golf and billiards, the player accumulates score points or tokens. The winning condition could be a simple target score or a combination of various score points for different game tasks.

The how-to-win mechanic is an important tool in attracting the player. An easy how-to-win mechanic can ruin even the best designed game.

TAKE NOTE*

A well-fought victory is better cherished than an easy one. Therefore, do not create win mechanics that are too easy to achieve.

GAME GOALS

CERTIFICATION READY
What are suitable goals for a fantasy game?
1.3

A game goal is the collective success criteria for the tasks in the game. Let us assume the following game mechanics for a game:

- **Quest:** Kill the opponent.
- **Task:** Shoot at the opponent using a sniper rifle.
- **Activities:** Aim at the opponent through the sniper hole and press the trigger.
- **Win:** Win happens when the player is able to kill his opponent without getting any fatal injuries.

For this game, the game goal will be:

To successfully complete all tasks and kill the opponent without getting any fatal injuries.

SKILL SUMMARY

IN THIS LESSON, YOU LEARNED:

- The idea for a game encapsulates defining the motivation for the set of players you are targeting. It also encapsulates deciding the genre and platform that is most appropriate for those players.
- People play games to satisfy their basic needs, which include quest, learning, task management, determination, competence, and thrill.

- The target audience for a game can be categorized as casual, hard-core, and intermediary players.
- Based on the gameplay, digital games can be categorized into various genres. The genres include education, action, adventure, sports, simulation, role-playing, fantasy, card, and board games.
- Games can be distinguished based on their online or offline status, number of players, and gaming platforms. Gaming platforms include console, PC, arcade, and mobile.
- A mission statement describes the main objective of the game, challenges to the achievement of that objective, and the resources available to players to help achieve that objective.
- The storyline of a game is the plot in the game and maps to the deepest needs of the players.
- A gameplay describes the various elements that bring about player interaction with the game.
- Game setting influences the look and feel of the game, such as the visual theme or cinematic of the game, and the objects, characters, user interface, audio theme, and any other element that is a part of the game.
- Cinematic means the sequences in a game over which the player has no or only limited control.
- Objects are the interactive or noninteractive visual elements of the game that complete a scene and make the game look believable.
- Characters in a game are defined based on the actions they perform and the things they say.
- A well-designed UI makes the game easier to play.
- To design effective game mechanics, you must first understand how the game works.
- Quests should lead to a reward mechanic in which the player is awarded some points for being successful.
- Task mechanics give the player a to-do list, which in turn can be broken down into specific activities.
- How-to-win game mechanics specify the conditions under which a player can win the game.
- Game goal is the collective success criteria for the tasks in the game.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. Achievement is a(n) _____ need of players that motivates them to play games. The Psychological game genre uses more than one player to be part of the gameplay.
2. _____ game genre involves spontaneous and quick challenges.
3. _____ is the primary device in card games.
4. _____ game genre emulates real-world activities.
5. A quest mechanic that requires players to kill an opponent is called _____.
6. A game's visual theme, user interface, objects, characters, and audio theme constitute the _____ of the game.
7. The _____ of a game define the interaction between the game and the players.

8. A player interacts with a game through its _____, which is an important component of gameplay.
9. A number of quests strung together form a(n) _____.
10. Activities are a subset of _____ mechanics.

Multiple Choice

Circle the letter or letters that correspond to the best answer.

1. You decide to develop a game in which the player can play the role of the characters in the game. Which of the following game genres does your game belong to?
   a. MMORPG
   b. RPG
   c. Adventure
   d. Simulation
2. You decide to develop a game in which the players can play using their PSP. Which of the following game platforms would you choose?
   a. Video game console
   b. Handheld game console
   c. PC game
   d. Arcade
3. You want to build a game on an open platform. Which of the following platforms would you choose?
   a. Video game console
   b. Handheld game console
   c. PC game
   d. MMORPG
4. You develop an online role-playing game for a large number of gamers in which the gamers can share the game resources. Which of the following platforms would you choose for your game?
   a. Video game console
   b. Arcade
   c. Mobile
   d. MMORPG
5. Which of the following are game mechanics elements to consider when conceptualizing a game? (Choose all that apply.)
   a. Quests
   b. Tasks
   c. Cinematics
   d. Game goals
6. Which one of the following should you consider while writing a storyline?
   a. Emotional bond with lovable, friendly characters
   b. Fetch quests
   c. Target player's motivational need
   d. Turnarounds
7. Which of the following qualities should the UI of a game possess to be called intuitive?
   a. The player can easily access information through the UI.
   b. The UI provides appropriate feedback when the player accesses various UI elements.
   c. The UI allows the player to find the required tools to perform the task at hand.
   d. The player can customize the UI as per his or her requirements.

8. Which of the following game mechanics will lead to a reward mechanic in which the player is awarded some points for being successful?
   a. Tasks
   b. Quests
   c. How to win
   d. Activities

9. "The player needs to cross a dangerous river using a boat and deliver the secret message to the army." Which type of quest is this?
   a. Kill
   b. Collection
   c. Target
   d. Secret

■ Competency Assessment

Project 1-1: Defining the Target Audience

You decide to create a game in the sports genre. You use a mobile device as your gaming platform. Define the target audience for your game with respect to the selected game genre and game platform.

Project 1-2: Defining the Visual Theme of a Game

You are creating an educational game for children to aid them in learning math through games. Define the visual theme for your game.

■ Proficiency Assessment

Project 1-3: Creating a Game Storyline

You are developing a fantasy game. Develop the storyline and game mechanics for your game.

Project 1-4: Selecting a Platform for an Online FPS Game

Your company develops a video game in the shooter genre. Keeping the recent trend in mind, they decide to build a game on the FPS genre. They plan to host it online to enable multiple audiences to play the game simultaneously. Choose the suitable gaming platform and explain the reason for your choice.

LESSON 2

Identifying and Managing Game Requirements

EXAM OBJECTIVE MATRIX

SKILLS/CONCEPTS	MTA EXAM OBJECTIVE	MTA EXAM OBJECTIVE NUMBER
Identifying Basic Game Requirements	Choose an input device.	2.1
	Choose an output device.	2.2
Identifying Game Performance Requirements	Work with the network.	2.3
	Manage game performance.	2.4
	Understand the different game platforms.	2.5

KEY TERMS

central processing unit (CPU)
Distributed Virtual Environment (DVE)
game controller
game performance
graphics card
graphics processing unit (GPU)
latency
load balancing
output resolution
profile
screen
Transmission Control Protocol (TCP)
User Datagram Protocol (UDP)
Web service

The first step before beginning the development of any software application is the identification of the basic requirements and the methods to manage these requirements. This implies identifying the type of hardware and network that the software application might use. The hardware requirement can be storage medium, input/output devices, and so on. The network requirement can be the type of protocol and the underlying network architecture. Identifying these requirements helps to speed up the development time because the requirements are clear and well understood. This step applies to the development of a game application as well. Before developing a game, the development team has to introduce themselves to the different input/output devices, graphics cards, and the performance of various platforms. Moreover, they also need to know the different network protocols and network architectures that their game might use.

■ Identifying Basic Game Requirements

THE BOTTOM LINE

The key task to perform before any development process is to identify the fundamental elements required for development. One of the fundamental elements required to be identified for a game development is the suitable input/output device for the selected platform.

A gamer's interaction with the game depends on two components:

- The type of input devices the gamer uses to interact with the characters of the game
- The type of output devices the gamer uses to see the visual output of the gameplay

The choice of input device should help the player to easily navigate and interact with the game. Similarly, the correct choice of output device can help reduce the risk of redesigning the visual elements or the graphics for the game. Mobile devices, such as cell phones and tablets, have smaller screen resolutions than a computer monitor. Therefore, before designing the graphic elements, you should consider the visual screen size for a game. The graphics designed for an output device/platform do not translate to other platforms. For instance, when a game designed for the personal computer (PC) is run on a mobile platform, the game characters might appear too small, leaving the player with a bitter experience of the game graphics.

Identifying the Input Device

CERTIFICATION READY
What are the commonly used input devices for video games?
2.1

The input devices for a game are accessories that enable the players to provide input to video games. The players use the input devices to control the objects or characters in the game. The commonly used input devices for video games are generally classified as ***game controllers***.

Today, a wide variety of game controllers are available in the market. Most of the game development companies develop games that support a wide variety of input devices. Some of the commonly used game controllers are:

- Control pad, Joypad, or Gamepad (wired or wireless)
- Mouse
- Keyboard
- Kinect
- Mobile devices
- Steering wheel
- Joystick
- Other input devices

CONTROL PAD, JOYPAD, OR GAMEPAD (WIRED OR WIRELESS)

CERTIFICATION READY
What is a control pad?
2.1

A control pad—also known as a joypad, gamepad, or direction pad (D-pad)—is a game controller, which usually comes along with video game consoles. An example of a control pad is an Xbox 360 controller.

www.itbookshub.com

The players can hold the control pad in their hands. They can use their fingers, especially their thumbs, to control the objects or characters in the game. The pad comes with a set of action buttons and a direction controller (see Figure 2-1). Players usually operate the action buttons with their right thumb and the direction controller with their left.

The advantage of the control pad is that it allows players increased control over the characters in the game. Moreover, it allows the players to use fewer fingers to play the game. Although the control pad usually supports the video game console, it is also available for PCs. You can connect a control pad to any of the USB ports on the video game console or PC through a wire or a cord. Wireless control pads are also available, which run on batteries. For example, an Xbox 360 controller is available in both wired and wireless versions.

Figure 2-1

A game system control pad

©Microsoft Corporation

MOUSE

CERTIFICATION READY
How is a mouse used in games?
2.1

The mouse is ubiquitous on PC games. The typical use of the mouse is for aiming and controlling the game. In other words, a mouse can be used when the game requires more precision, such as in shooter games. The game genre where the mouse is more suitable includes first-person shooter (FPS) games, card games, and board games.

The advantage of using a mouse is that players can handle the mouse with ease and have better control over the object in the game. However, the main action a common mouse supports is a click. In complex games, a player can perform many actions, such as jumping, hitting, running, walking, turning, speeding, and so on. Every action cannot be associated with the click of a mouse. Gaming mice attempt to overcome this limitation by building in additional buttons that are user-programmable to perform a variety of game-specific actions. An example of game-specific mouse peripheral is the Microsoft Sidewinder Gaming Mouse (see Figure 2-2).

Figure 2-2

The Microsoft Sidewinder gaming mouse

©Microsoft Corporation

KEYBOARD

CERTIFICATION READY
What are gaming keyboards?
2.1

Keyboards are ubiquitous on PC platforms and most users are comfortable using them. The large number of unique keys and key sequences gives players a broad palette of commands for fine control over game objects.

Like mouse peripherals, there are also several keyboard peripherals that are specifically designed for playing video games. These keyboard peripherals are built with game-specific functions. An example of a game-specific keyboard peripheral is the Microsoft Sidewinder Gaming Keyboard (see Figure 2-3). Few games are designed to require these keyboards.

Figure 2-3

The Microsoft Sidewinder gaming keyboard

©Microsoft Corporation

KINECT

CERTIFICATION READY
What is Kinect?
2.1

Kinect is the latest addition to game controller peripherals (see Figure 2-4). Kinect works with Xbox 360 and is a motion-sensing device that senses the motion of the players using a camera and translates them into the motions of the character in the game. For example, to play a game of tennis, the player needs to move the hand as if hitting the ball with the racket, and the character does the same action.

Mostly sports games such as soccer, rugby, shooting games, and simulation games use Kinect as their input device. You can use Kinect when you want the players to get physically involved in one of the game characters and use their force and effort on the character. Microsoft designed Kinect for Xbox 360 to widen its normal gamer base. A Windows version of Kinect is also available.

The advantage of using Kinect as an input device in games is that it helps the player to be involved physically with the game's character. However, the disadvantage is that Kinect works only for the Xbox 360 console and Microsoft-based computers.

Figure 2-4

The Kinect input device with an Xbox console

©Microsoft Corporation

MOBILE DEVICES

CERTIFICATION READY
How is a mobile device used for gaming?
2.1

You already learned in Lesson 1, "Ideating and Conceptualizing a Game," that you can use mobile devices as a gaming platform. Alternatively, you can also use the mobile device—such as the Windows Phone—as an input device for the Xbox 30 console (see Figure 2-5). Players can use the Windows Phone to control and interact with an Xbox 360 game. This is possible by using the Xbox Companion app available in Windows Phone. Using the app, players can control video games on the Xbox 360 console. Moreover, players can access games available on Xbox Live using Windows Phone. Recently, Microsoft announced the Xbox SmartGlass app for mobile devices. The Xbox SmartGlass app allows mobile devices, such as tablets or phones, to connect with Xbox 360. As a result, players can use the mobile device as another controller for the game.

Figure 2-5

An example of a Microsoft Windows Phone 8

©Microsoft Corporation

STEERING WHEEL

CERTIFICATION READY
What is the role of a steering wheel in games?
2.1

A steering wheel is a game controller that has a large round wheel with one or more fire buttons (see Figure 2-6). The fire buttons are simple switches that help the gamer to trigger some type of action. The player can control the movement of the game character or the object in the game by steering the wheel. Many wheels can turn a maximum of 270 degrees lock to lock. Some wheels from Logitech can turn 900 degrees lock to lock. Most of the racing arcade games and racing simulator games, such as *Richard Burns Rally*, support steering wheels.

An advantage of a steering wheel game controller is that it can be used in all types of racing video games, where the gamer takes part in a racing competition with any type of vehicle. Moreover, steering wheels provide the gamer with better immersion when steering a virtual vehicle. For example, in a car racing game, you can use the steering wheel to accurately manipulate the steering angle to manage the car. The disadvantage is that apart from racing video games, other game genres do not support steering wheels.

Figure 2-6

A steering wheel game controller

©Microsoft Corporation

JOYSTICK

CERTIFICATION READY
How is a joystick used in games?
2.1

As the name suggests, the joystick is like a stick that the players can use to control the game (see Figure 2-7). The device allows faster maneuvering of the objects in the game. Players can tilt the joystick within a cone around the resting axis of the stick; the controller reports the direction and deflection angle to permit the user interface to respond. Players can operate the joystick with their primary hand by holding the base in the opposite hand or placing it on a desk. Flight simulator games often use a joystick.

The advantages of a joystick are that you can use this game controller for all the commonly used game platforms, such as video game consoles, arcade machines, and PCs. The disadvantage is that it is difficult to use a joystick when precision is required while selecting game objects on the screen.

Figure 2-7

A joystick game controller

©Nikita Rogul/iStock Photo

OTHER INPUT DEVICES

Apart from the standard input devices, numerous additional input devices are available for different games. The following list briefly describes some of these input devices:

- **Light gun:** This device helps to aim and shoot at the targets on a game screen. This device uses light to detect the targeted location on the game screen.
- **Dance pad or dance mat:** This device helps provide input to dance games. The device consists of flat pressure-sensitive gamepad buttons set on a mat. The players step on the mat and press the action buttons to control the game.

- **Wii balance board:** This device is used for the Wii and Wii U video game consoles. The players stand on this unique platform and perform various physical exercise activities.
- **Musical game controller:** This input device is used in music-themed video games. This controller resembles musical instruments, such as guitars, drums, and keyboards.
- **Motion sensing device:** Like Kinect, other motion sensing devices are available for different game consoles. For example, Nintendo's Wii system uses a Wii remote. The Wii remote allows the players to interact with the video game through gestures and pointing. The device has an accelerometer to recognize the gesture and an image sensor for pointing. Another example of a motion sensing input device is PlayStation Move from Sony.

Identifying the Output Device

An output device for a game is an electronic device, such as a television or a monitor, which displays the visual output of the game.

Output devices fall into two broad categories, namely, display devices and sound devices. The display devices, as the name suggests, display the visual element of the game, and the sound devices, such as speakers or headphones, provide the audio output of the game.

DISPLAY DEVICES

CERTIFICATION READY
How do display devices influence game output?
2.2

A wide variety of display devices is available to display the visual output of games. The choice depends on the following criteria:

- **Genre of game:** Games of different genres require different visual output devices. For instance, for a shooting game, the output device can be a simple display unit of any kind, such as a monitor or television. For mobile games, the display device is the screen of the handheld device, typically smaller than a 7-inch diagonal.
- **Game platform:** Different game platforms require different display devices. For instance, games made for mobile platforms with a resolution matching to the handheld devices are typically not designed to be played on a larger screen, such as a projector screen, because the graphic tends to pixelate. However, the new generation of devices, such as the iPad and Samsung Galaxy Note, do support a high resolution of display.
- **Game graphics:** The graphics of the game also play a major role in choosing the display devices. Games that use heavy graphics cannot be displayed on devices with low-end hardware that does not support enhanced graphics. Moreover, games with small characters cause visibility issues in certain platforms hindering the gameplay. For example, the *Age of Empires* game is a highly graphical war strategy game. It requires a detail level of game graphics that is not well suited for the small size of mobile device displays. The small size of the game characters causes a lot of hindrance for the player while playing the game.
- **Target audience:** Another criterion that you need to consider when selecting the visual output device is how often the target audience uses those output devices. You need to ensure that the target audience is comfortable playing the game and this can happen if they can use a familiar visual output device.

Three commonly used display devices are:

- Televisions and monitors
- Handheld devices
- Touchscreen devices

TELEVISION AND MONITOR

CERTIFICATION READY
Why are different games more appropriate for different display types?
2.2

Almost all homes have a television. Players can connect their gaming device, such as their video game console, to the television set and view the visual output on the television display unit.

Similarly a monitor, also referred to as a ***screen***, is an electronic display unit for computers. In general, games played on the PC use the monitor for display.

A variety of display units are available for both televisions and monitors. Some of them include:

- **Liquid crystal display (LCD):** LCD screens are thinner and lighter than the standard television. They use LCD display technology and are available in larger sizes.
- **Plasma display panel (PDP):** A plasma display panel displays bright images and uses a wide color range. The panels are available in 30 inches or larger sizes.
- **3D:** Today, most of the high-end televisions and monitors come with 3D display and viewing devices, such as 3D glasses, to view the images in three-dimensions. To enhance the gaming experience, many gaming companies are creating games specifically for 3D output devices. A player has to wear 3D goggles to play the game.

Now people are shifting from the standard television to high-end televisions, such as the high definition LCD/LED television (see Figure 2-8). Keeping this in mind, developers create suitable graphics for their games to adapt to the present-day demand. Almost all games support television or monitor screen as the output device, except mobile games.

Figure 2-8
A high definition LCD display

©Vitalina Rybakova/iStock Photo

HANDHELD DEVICES

CERTIFICATION READY
How does the display quality of a handheld device differ from that of a PC or a console?
2.2

Handheld devices include various electronic gadgets, such as a mobile phones, smartphones, tablets, PDAs, and even handheld game consoles. Unlike computer systems with monitors or consoles on televisions, these handheld devices provide a relatively small display size. They support low-end graphics compared to a PC or a console. However, good quality handheld consoles can provide an enriching gaming experience.

People who travel a lot and want entertainment during transit prefer handheld devices. Games designed for these devices use simple graphics due to their small display size and processing power as compared to PCs and consoles.

The advantages of using handheld output devices are primarily portability, simplicity, and ease of availability. Despite its growing popularity, handheld devices, due to their small

display size, can support only rudimentary graphics, severely limiting the variety of games that can be made available on these devices. However, the number of handheld devices has increased the demand for more games in this platform, which in turn is driving new technology adoption for better graphic support.

The screen size and display quality of handheld devices are increasing rapidly, and although the devices are now capable of displaying higher resolution graphics, the small screen size is still an important factor to consider when choosing your system and device.

TOUCHSCREEN DEVICES

Touchscreen devices allow users to interact with the computer or game using their fingertips. Touchscreen kiosks, such as the passenger information systems at train or bus terminals, are designed for self-service installations. Today, game makers are harnessing the power of these touchscreen kiosks to provide simple games that players can play using their fingers.

The advantages of choosing a touchscreen kiosk as an output device is the availability of the game to a larger number of people, ease of playing, and simple entertainment on the go. However, not all games support touchscreen kiosks.

Touchscreen technology is also more common in mobile devices, such as smartphones and tablet computers. Therefore, a wide variety of touchscreen games is available for the mobile platform. There are also versions of desktop and laptop computers with touchscreen displays. Figure 2-9 shows a picture of a device with touchscreen technology.

Figure 2-9

A touchscreen device

©Mihai Simonia/iStock Photo

SOUND DEVICES

CERTIFICATION READY
How do sound devices influence games?
2.2

Sound plays an essential part in many gaming experiences. Without sound, the feel of the gaming environment is incomplete and players would not be able to immerse themselves fully into the gaming world. For instance, imagine you are playing an adventure game. When the game character walks through the forest, you can hear the sound of dried leaves crinkling underfoot. This gives you the feeling that the character is actually walking through a forest, and this effect enhances your gaming experience.

Many games require sound to provide a feel of the game environment and to get the player engaged in the game. Furthermore, sound creates the mood of the game.

You can choose to have local speakers or opt for a complete surround system for a better gaming experience (see Figure 2-10).

Figure 2-10

A complete surround sound system

©Baris Simsek/iStock Photo

Table 2-1 lists some sound devices and describes their advantages and disadvantages.

Table 2-1

Advantages and Disadvantages of Sound Devices

Sound Device	Advantages	Disadvantages
Headphones	• Provide privacy • Prevent disturbance to others • Can be used for games that support 3D positional audio	• Increase the chances of hearing impairment
5.1 speakers	• Provide optimum sound quality • Provide life-like reproduction of sound • Provide better gaming experience	• Expensive
2.1 speakers	• Most commonly used • Affordable • Available in a wide range of products	• Provide lower quality of sound
Standard speakers	• Affordable	• Provide lower quality of sound

■ Identifying Game Performance Requirements

THE BOTTOM LINE

To create a player-friendly game, you must identify the hardware, memory, and networking requirements for an optimal ***game performance***.

CERTIFICATION READY
What are the ways to manage the performance of the TPS strategy game?
2.4

You can judge the game performance by analyzing how well the game can run on a chosen device with specific graphics. You also need to analyze whether the game can adapt to the network setup when players want to play the game online.

The game designer needs to understand the importance of game performance for a good gaming experience. For example, an FPS game provides a richer experience on a PC than

it does on a mobile device because of the availability of new age ***graphic cards*** to boost the processing power on PCs. So the difference in platforms on which the game is played is an important factor that affects the performance of the game. The game designer must therefore choose the game platform carefully.

Usually, if you want the game to work on all platforms without any performance issues, you would need to compromise on the graphics quality. However, rapid development in technology is helping to overcome the differences in platforms.

TAKE NOTE*

Throughout this book, we use "fps" to refer to frames per second, and "FPS" to refer to first-person shooter.

The game that you create performs at its best when it runs with a good frame per second (fps)—above 45 fps—on the target hardware platform. It is therefore vital to effectively manage the performance requirements of the game.

Managing game requirements includes performing several tasks, such as understanding what exactly is meant by game performance and identifying the hardware, memory, graphics, and networking requirements for the game to run without hindrance. It also includes ensuring that these requirements are satisfied by the game.

To effectively manage the game performance, you must manage three components:

- Platform-specific memory requirements
- Graphics performance requirements
- Networking requirements

Usually, the game code takes up some memory space. The memory that is available for the game code depends on the platform for which you are developing the game. For example, when you create a game for a console platform, the game code can run into gigabytes. However, a mobile platform has limited memory space and will not be able to handle a game with so much game code. Therefore, a game meant for the mobile device must have less game code than that for a console game.

Similarly, if you use heavy graphics and lengthy coding in your game, you run the risk of overloading the memory and causing the game to lag. This eventually leads to player dissatisfaction.

Managing Platform-Specific Game Requirements

Managing game performance requirements essentially refers to choosing a configuration for the game that will allow the game to perform properly without any interruptions.

Games are created to run on specific game platforms, such as consoles, mobile devices, and PCs. The games are loaded into the memory when running. Insufficient memory affects a game's performance. Therefore, the memory available in these platforms is an important factor to consider when developing a game. Mobile devices have a smaller expandable memory size than consoles or PCs. Therefore, the games developed for mobile devices must have less code than the games developed for consoles or PCs. Today, the games designed for the console platform are available on Blu-ray discs, CDs, or DVDs and can be several gigabytes in size. At the same time, games for the mobile platform are comparatively smaller.

CONSOLE

CERTIFICATION READY
How does the memory requirement of different console devices affect game performance?
2.5

Certain types of consoles, such as Xbox and PlayStation, have features that can accommodate a wide range of games. However, other types of consoles, such as Nintendo and Atari, might not have certain features, which limits the type of games that can be played.

For example, Xbox and PlayStation have both a built-in hard drive and external memory. This means that you can copy the game to the console or directly play the game from an external source, such as CDs, DVDs, and Blu-ray discs. However, Nintendo Wii U does not have a built-in hard drive and cannot play games that require built-in storage.

MOBILE

CERTIFICATION READY
How does the memory requirement of mobile devices affect game performance?
2.5

Mobile gaming is becoming more popular. Games for mobile devices have lower-end graphics than games for PCs or consoles. In general, mobile devices have small volatile memory. However, the new age smartphones and the new handheld console devices are rapidly overcoming the limited-memory hurdle. Most of the latest mobile devices can support an external memory card, which allows the user to play graphic-intensive games. Game response depends to a greater extent on RAM. Game assets can be stored in persistent memory, such as memory cards or built-in equivalents.

The possibility of using external memory presents new opportunities. Games developed for school-age children generally use rich colors, graphics, and heavy coding. In the past, mobile platforms did not have the hardware required for such high-memory-requirement games. However, today, with the improved hardware technology, you can also consider mobile platforms. A handheld console or a mobile device that has facilities to play an external source—such as a mini disc, mini cartridge (though not popular now), and SD cards—can be a good platform for games meant for school-age children. In addition, school-age children can easily operate smartphones. These devices can support graphics far better than normal mobile phones.

However, external memory does not always provide the perfect solution. Unlike smartphones and the latest handheld devices, mobile devices that can only read from a memory card severely limit the kind of games that can be played on the device. This is because the card provides only storage memory and not the memory required for running or processing the game. The memory required for running the game changes depending on the kind of mobile device or smartphone that the player uses. Therefore, you need to carefully study the suitability of the selected mobile platform for your game.

PERSONAL COMPUTER

CERTIFICATION READY
How does the memory requirement of PCs affect game performance?
2.5

PC users have a wide variety of games from which to choose. Users can easily upgrade the storage and memory required to run and process the games on this platform. Some games require the installation of a graphics coprocessor, if one is not available in the specified computer. However, new machines have an integrated graphics coprocessor.

Today, there are gaming computers available that can play high performance video games. A gaming computer is a PC that is specifically designed for intense gaming.

Table 2-2 presents the three main platforms and their advantages and disadvantages in terms of game performance.

Table 2-2
Comparison of Platform-Specific Gaming Performance

Gaming Platform	Advantages in Game Performance	Disadvantages in Game Performance
Console	• It is the most preferred platform. • It is the most suited with adequate memory.	The built-in storage memory is not easy to upgrade.
Mobile	• It is rapidly gaining popularity. • Games with low graphics can be played easily. • Latest handheld devices provide facilities to store games on external sources.	The kind of games that can be played on this platform is limited because of limited memory. An upgrade is not possible for memory required to run and process the games.
PCs	• Memory requirements can be easily met by enhancements.	It requires a heavy investment for the required high-end memory and graphics card.

www.itbookshub.com

Managing the Impact of Graphic Performance

Although different platforms impose specific restrictions on the performance of the game, the different hardware and software used for displaying the graphical outputs also impact game performance.

While playing games, a player might often experience a slowdown of the visual output of the game on the screen. This might be a sign that the player's graphics card is inefficient or does not support the graphics of the game. This, in turn, might affect the player's overall gaming experience.

To ensure that such situations do not arise, as a game developer, you should understand the impact of the hardware that your game might run on and manage your game's graphics performance accordingly. You should also consider the impact of the network on your game's performance, if your game runs on a network.

The following list of elements can influence the performance of the graphics used in your game:

- Central processing unit (CPU)
- Graphics processing unit (GPU)
- Reach
- HiDef
- Network impact

CPU vs. GPU

CERTIFICATION READY
What are the differences between CPU and GPU?
2.4

A ***central processing unit (CPU)*** is a hardware component inside the PC that performs all the processing activities of the system. The primary function of the CPU is to carry out instructions of a software program. This function can include various tasks, such as handling data, browsing the Internet, and playing music. In the early computing years, computer games used mild graphics and the CPU did the graphics processing by itself. Today, because of the advancement in video games, CPUs use a video card, also known as a ***graphics card***, to enhance the graphics processing.

A video card enhances the richness and the display of graphics on a screen. It contains its own processor known as the ***graphics processing unit (GPU)***. A GPU is a specialized processor that performs graphics processing rapidly. It is exclusively designed for depicting 3D graphics and drawing 2D images. GPUs are available in two forms: dedicated graphics cards and integrated graphics solutions.

In the integrated graphics solutions, the GPU is integrated into the computer's motherboard, and it uses portions of the CPU's main memory for graphics calculations. This helps reduce the capability of the GPU as it shares the CPU memory. However, the integrated graphics solution is cheaper and, therefore, most computers, such as the Intel HD, come with an integrated graphics solution.

TAKE NOTE*

Apart from PCs, GPUs are commonly used in mobile phones and video game consoles.

In contrast to the integrated graphics solution, in a dedicated graphics card the GPU comes with the card and has its own memory, which helps the GPU to perform the graphics calculations faster.

Although many games can be played using integrated graphics solutions, games that require heavy graphics processing provide better performance when played using a dedicated graphic card.

REACH VS. HiDef

CERTIFICATION READY
What are Reach and HiDef?
2.4

A graphic displayed on the screen is made up of pixels or dots. The number of such pixels or dots that make up an image is called the ***output resolution***. The larger the resolution,

the sharper the picture will be. The display resolution plays a key role in the game's performance. If the correct resolution is not used, the graphics might not be displayed properly, which will affect the game's performance.

Until sometime ago, output resolution was dependent on the platform chosen. To manage the output resolution and, in turn, ensure the best quality of graphical performance, game developers had to create the required platform-dependent code and test the code for each platform. To ease this task for developers, Microsoft XNA Studio 4.0 Framework uses Profile. A ***profile*** is a set of features employed in the hardware and is platform-independent. This allows the game code written for one platform, such as a PC, to run on a different platform, such as the Xbox 360 console, with little or no change in the code. This is possible because the application programming interfaces (APIs) that access the profile in the hardware are consistent across platforms. You can set the profile during the design time or at run time.

Profiles are of two types, namely Reach profile and HiDef profile.

TAKE NOTE*

DirectX is from Microsoft and is a set of APIs that handles high performance multimedia applications including games. DirectX 9 is the advanced version of DirectX. DirectX 9 GPU card is a video card that is compatible with DirectX 9 features.

TAKE NOTE*

Games with HiDef profile can also run on Windows-based computers that have DirectX 9 GPU card, if the DirectX 9 card implements named DirectX 10 features.

REACH PROFILE

The Reach profile comes with a limited set of graphics features and capabilities. It is integrated in the hardware of almost all platforms. You can specifically use the Reach profile if you are creating games for Windows-based PCs that use the DirectX 9 GPU card, XBOX 360 consoles, and Windows Phone. The main advantage of using Reach is that you can create games for various platforms much faster.

HiDef PROFILE

HiDef profile has the highest level of graphics features and is suitable for platforms with hi-powered hardware and enhanced graphics capabilities. You can use HiDef profile if your games have rich graphics and you want your games to run on Xbox 360 consoles and Windows-based PCs that have DirectX 10 GPU card.

MORE INFORMATION

To know more about the specific features supported by Reach and HiDef, refer to the Microsoft Developer Network (MSDN) site.

NETWORK IMPACT

The network also influences the graphics performance of games. There are two types of network, online and offline.

A game created for an online network uses the Internet. It is the trickiest form of game distribution. Different countries support different Internet bandwidths and different people use Internet connections with different bandwidths. Therefore, creating a game that can easily work on the lowest bandwidth is a huge task, and the problem is complicated when the number of users who join the game is also large. Thus understanding the maximum limitation of the user who joins a level plays an important role in online games. To create a game for a large group of people, you need to create the game code and graphics in such a manner that it is light and easy to run on the server so that the flow of the game is not interrupted.

Games created for offline networks use internal networks, such as an organization's LAN or arcade. The number of people playing offline games is not large (it can even be one player) and the bandwidth of the network remains consistent. Therefore, you can use rich graphics for offline games. Depending on the platform and the distribution of the game, whether online or offline, a game company must consider different requirements. Table 2-3 lists the requirements for online and offline games.

Table 2-3

Requirements for Online and Offline Games

Online Games	Offline Games
• Use light graphics and code • Use a dedicated server • Ensure that the game is able to perform well on low bandwidth • Ensure that the game is able to handle a large number of users	• Can use high-end graphics and game code • Do not require a dedicated server • Ensure that the game is able to perform well on low bandwidth • Ensure that the game is able to handle a large number of users

Managing Network Requirements

Managing the network is one of the essential tasks to provide the players with an enhanced online gaming experience.

Today, online games are very popular. These games can be played on the Internet with players from all around the world. To ensure that the online games work smoothly, you must select the right code, graphics, and server. If the selected code, graphics, or server is not proper, the game might not get the required resources to run smoothly. This can lead to deterioration of performance, and the game—and the server on which it is hosted—might eventually stop responding.

UNDERLYING NETWORK ARCHITECTURE

When it comes to online games, today the ***Distributed Virtual Environment (DVE)*** has become a popular network application. A Virtual Environment (VE) is a computer-generated simulation in which the system simulates a 3D virtual world. When a VE supports multiple interacting users and is distributed across several computers connected by a network, it is called as a *DVE*.

Today, multiplayer online games use DVE, where gamers share the VE. Independent users share the VE through their avatars. The user can control the state of the avatar through the client machine. A large number of players can connect to the DVE system using their client computers through different networks, such as a LAN, a network of computers in a home or an organization, or a Wide Area Network (WAN) like the Internet.

The following are the different underlying architectures that support multiplayer online games:

- **Centralized-server architecture:** DVEs based on this architecture contain a single server and multiple clients are connected to the server. In this case, the server manages the entire virtual world. As a result, if the number of avatars or client computers increases, the server becomes a potential bottleneck. Therefore, DVEs based on this architecture support a fewer number of clients or avatars.
- **Networked-server architecture:** DVEs based on this architecture contain multiple servers and each avatar or client computer connects exclusively to one of these servers. Because this architecture is distributed over several servers, it is robust, scalable or expandable, and flexible. However, it requires efficient ***load balancing***, which means distributing the workload across several computers to avoid overload and to gain the best resource utilization. Load balancing ensures that all client computers get an equal share of resources at the servers.
- **Peer-to-peer architecture:** DVEs based on this architecture have the client computer playing the role of the server. This helps the highest level of load distribution.

Although DVEs based on networked server architecture are scalable, they tend to increase the network traffic. This in turn affects the number of avatars supported. A peer-to-peer architecture is the best suited for online games that support a large number of players because there is a huge amount of interactivity. Moreover, as DVEs are network-based applications, basic network management also becomes an essential part.

NETWORK MANAGEMENT

CERTIFICATION READY
What is the basic network management involved in gaming?
2.3

Basic network management with respect to online games involves the following:

- **Managing the network operations:** You need to ensure that the network, network devices, and the network services are available and are performing smoothly. You also need to monitor the network constantly to identify any network issues as soon as they crop up, so that the issues can be rectified immediately and the users are not affected.
- **Administering the network:** You need to keep track of all the network resources. You need to ensure that the resources are correctly configured and assigned, and that the resources are secured against malfunctioning or hacking.
- **Maintaining the network:** You need to perform maintenance tasks, such as repairing failed or malfunctioning network resources, upgrading the network resources according to the requirements of the network and the usage of the network, and taking preventive and corrective measures to ensure that the network provides optimum performance.
- **Provisioning the network:** You need to ensure that the network resources, including the network devices and services, are configured correctly so that they provide the expected level of features and performance.
- **Network traffic:** You should measure the network traffic requirements in terms of bandwidth or the rate of data transfer required by both servers and clients to communicate with each other within a specific period. Determining the bandwidth accurately in terms of bits (of data) per second or bps helps the network designers to design a scalable system that effectively supports multiplayer online games.
- **Latency:** The term ***latency*** refers to the time required for data to travel from one computer to another computer. Several elements can cause high latency, including network congestion, interference on lines, and improperly configured games. Latency also depends on the type of network carrier, such as cable, DSL connection, or satellite connection.

Apart from the previous tasks, you should also decide on the type of the communication protocol, such as the TCP and UDP, which your network might use for communication between the networked computers.

TCP AND UDP

CERTIFICATION READY
How are TCP and UDP useful in games?
2.3

Let us begin with understanding what the Internet Protocol (IP) is. It is a protocol that sends packets of data, called datagrams or network packets, from one computer to another. To send data from the source computer to the destination computer, the IP passes the data from one computer to another until it reaches the destination computer. It is similar to passing on a handwritten note from one person to another to get across to another person standing at the other end of a crowded hall. You pass on the note to the nearest person, who in turn passes it to another person and so on until it reaches the intended person. During this passing, the note could fall down and get lost. In addition, the note might not reach the intended person; on the way, another person could just take the note without passing it on to the next person. This is also the case with IP. There is no guarantee that the sent data will reach the correct destination.

Now let us look at another core protocol of the IP family, ***Transmission Control Protocol (TCP)***. The TCP is built on top of IP and is commonly referred to as TCP/IP. It adds a lot of features and complexity to IP. As the name suggests, TCP/IP defines or controls how data is transmitted from one computer to another. At the source, it groups the data to be transmitted

into continuous streams of bytes and passes it on to IP for transmission to the destination. Moreover, TCP ensures that the packets reach the destination by adding sequence numbers and timeouts to each byte that is transmitted. It then expects a positive acknowledgment from the destination within the timeout interval. If it does not receive an acknowledgment, TCP resends the data. At the destination, the receiving TCP uses the sequence number to rearrange the packets, and it removes the duplicate packets. In this way, TCP/IP guarantees that each packet reaches the destination and is not lost in transit. Therefore, TCP/IP is considered a reliable protocol. It is not surprising that TCP/IP controls most of the activities that you perform online, such as browsing the Web, sending and receiving e-mail, chatting using instant messaging, transferring files, and performing remote computer/network administration.

Another protocol of the IP family is ***User Datagram Protocol (UDP)***. UDP does not add a lot of features to IP like TCP does—it just forms a thin layer over IP. So, as with IP, data is sent from the source to the nearest computer and passed between numerous devices before it reaches the destination. There is no guarantee that the data will reach the intended destination. There is also no guarantee that the data will reach the destination in the same order that it was sent. The only guarantee that UDP provides is that the data packet sent from the source will either reach the destination as a whole or will not reach the destination at all. UDP will not deliver broken packets. Also, UDP is faster than TCP because there are no additional checks or features.

Depending on the game, a developer can choose the type of protocol. A twitch game might involve UDP, whereas shared state simulations might be better handled with TCP.

SETTING UP WEB SERVICES

CERTIFICATION READY
How are Web services useful in gaming?
2.3

Today, online games are slowly moving to an e-business environment in order to develop a sustainable business model. However, the e-business environment should provide a robust and efficient business model to the game publishers, game developers, players, and the service provides. To cater to this need, many online games are integrating their game application with Web services technology. For example, Microsoft Xbox Live uses a collection of Web services for game publishers and game developers to access game data. The Web services include managing accounts, creating competition, and much more. This enables the game publishers to extend the reach of their games to the Web community. Moreover, Web services are also used to manage persistent online games, such as MMORPG games.

A ***Web service*** is a software application that exchanges information using standard protocols. It helps computers on any platform to exchange information over intranets (a network of computers within an organization using IP for transmission of data), extranets (private networks that are extensions of an intranet for the purpose of sharing information to specific users), and across the Internet by providing secured and reliable messaging. A Web service uses XML for representing data, Simple Object Access Protocol (SOAP) for data exchange, and Web Services Description Language (WSDL) for describing the service's functionality. You can create and set up Web services for your game using Microsoft Visual Studio.

SKILL SUMMARY

In this lesson, you learned:

- Game controllers help the players to interact and control the gameplay.
- Commonly used game controllers include mouse, keyboard, Kinect, control pad, and mobile devices.
- Display devices and sound devices help the gamer experience the visual output and audio output of the game.

- Game performance is a measure of how well the game can run on a chosen device with specific graphics and how well the game can adapt to the network setup when players want to play the game online.
- A game performs at its best when the game runs without any interruptions on the lowest of the configuration setup.
- Managing game performance requirements essentially refers to choosing a configuration for the game that will allow the game to perform properly without causing any interruption to the player during the game.
- The memory available for playing games depends on the platform selected for the games.
- A video card enables the output of images on the screen and it uses GPU for processing the graphics.
- A dedicated graphics card has a GPU embedded in the video card and has its own memory.
- In the integrated graphics solutions, the GPU is embedded in the CPU and it shares the CPU memory.
- A profile describes a set of features integrated in the hardware, which permits game developers to create platform-independent code. Reach and HiDef are the two types of profiles.
- The Reach profile supports limited graphics features and is implemented in almost all platforms.
- The HiDef profile supports the highest level of graphics features and is implemented only on platforms with high-end hardware.
- A Virtual Environment (VE) is a computer-generated simulation in which the system simulates a 3D virtual world.
- A Distributed Virtual Environment (DVE) supports multiple interacting users and is distributed across several computers connected by a network.
- Multiplayer online games use DVE.
- The underlying network architecture of DVE includes centralized-server architecture, networked-server architecture, and peer-to-peer architecture.
- TCP/IP is a reliable protocol that controls the transmission of data over the network.
- UDP is faster than TCP/IP because UDP does not provide additional checks or features to the data packets.
- A Web service is a software application that exchanges information using standard protocols and provides interoperability.
- Many online games integrate their game application with Web services technology.
- Microsoft Xbox Live uses a collection of Web services for game publishers and game developers to access game data.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ peripherals help the players to interact and control the game play.
2. _____ output devices have small screen displays.
3. _____ game helps the players to control the player character without touching the game controller.
4. Kinect works on the _____ console.
5. _____ input devices support low graphics hardware.
6. _____ and _____ are the two types of profiles.

7. _____ Protocol does not perform checks on data packets.
8. _____ GPU has its own memory.
9. _____ includes an error correction mechanism in the packets to ensure that the data is not distorted when it reaches the destination.
10. A flight simulator game often uses a(n) _____ as an input device.

Multiple Choices

Circle the letter or letters that correspond to the best answer.

1. You decide to develop a game for people who travel a lot. Which of the following output devices should you choose?
   a. Handheld
   b. TV
   c. Monitor
   d. Plasma display panel

2. You are developing a shooting game. You would like to provide players with a rich gaming experience by physically involving them in the main player character of the game. Which of the following input devices should you choose?
   a. Console
   b. Joystick
   c. Kinect
   d. Joypad

3. You create a HiDef game with rich graphics. Which of the following hardware settings would you recommend for your audience?
   a. DirectX 9 GPU
   b. DirectX
   c. DirectX 10 GPU
   d. All of the above

4. You are creating an online game. The underlying network architecture has one dedicated server and multiple clients connected to the server. Which of the following architectures does your network use?
   a. Centralized-server architecture
   b. Networked-server architecture
   c. Peer-to-peer architecture
   d. Client-server architecture

5. Your game runs on a variety of platforms. Which of the following should you choose to make your game interoperable?
   a. Reach profile
   b. HiDef profile
   c. Both Reach and HiDef profiles
   d. Write platform-specific code and check the code for each platform

6. You are developing an online game. Which of the following applications should you recommend to provide a sustainable business model?
   a. Web service
   b. SOAP
   c. XML
   d. WSDL

7. You are choosing handheld devices as the output device for your game. Which of the following fall under the handheld devices category?
   a. Tablet
   b. Television
   c. Video game console
   d. Monitor

8. You develop a flight simulator game for platforms that include a video game console, a PC, and arcade machines. Which of the following input devices best suits your game and game platform?
   a. Steering wheel
   b. Joystick
   c. Game-specific mouse
   d. Kinect

9. Your game characters are small, but the detailing of graphics is elaborate in your game. Which of the following output devices would you *not* recommend for your game?
   a. Mobile phone
   b. Television
   c. HD LED/LCD TV
   d. HD LED/LCD monitor

10. You are creating a racing simulation game. Which of the following input devices is best suited for your game?
    a. Steering wheel
    b. Joypad
    c. Game-specific keyboard
    d. HD LED/LCD monitor

■ Competency Assessment

Project 2-1: Using Screen

You are developing a third-person shooter (TPS) game like *Binary Domain*. Explain the different types of screens, their advantages and disadvantages, with respect to your game.

Project 2-2: Using TCP/UDP

You are developing a third-person shooter (TPS) game for online gamers. Explain the pros and cons of using TCP and UDP as the communication protocol for the network.

■ Proficiency Assessment

Project 2-3: Using Kinect and 3D Monitors

You decide to use Kinect and 3D monitors as the input and output devices for your game. Conceptualize a virtual simulation game that best suits the chosen devices.

Project 2-4: Using Reach and HiDef

You decide to use Reach and HiDef profiles to create a platform-independent game. Explain the pros and cons of using Reach and HiDef profiles with respect to all the commonly used output platforms.

Creating the Game Output Design

LESSON 3

EXAM OBJECTIVE MATRIX

SKILLS/CONCEPTS	MTA EXAM OBJECTIVE	MTA EXAM OBJECTIVE NUMBER
Creating the Visual Design	Design the user interface. Draw objects.	1.4 3.3
Deciding the Output Parameters	Understand rendering engines.	3.1

KEY TERMS

bitmap
codec
compression
DirectX
display initialization
display mode
graphics type
lossless compression
lossy compression
parallax mapping
rendering engine
resolution
UI components
UI concept
UI layout
Vertical Synchronization (VSync)
visual design
visual design elements

John Smith works at Contoso Game Studio as a designer. At a recent product launch meeting, his development team informed him of a new product they are working on. The development team has created a new first-person shooter (FPS) game for console and computer systems. It is now the job of John and the design team to design the game's visual world.

In the design phase, John and his team think about two main items—the visual design of the game and the output parameters. These two items constitute the game output design, which controls how the game will look, feel, and run. A good game output design makes the game run seamlessly and creates a virtual world for the player.

This phase of the game creation impacts the game functionality on many levels. In this case, to create the visual design, the team must first select the appropriate type of graphics—two-dimensional (2D) or three-dimensional (3D). Based on the graphics type, the team creates

the visual design elements and the User Interface (UI) design. Choosing the output parameters beforehand helps ensure that the design elements integrate seamlessly into the game. It also helps ensure that the game runs smoothly in the required resolution and as per the basic performance requirements identified for the game.

■ Creating the Visual Design

THE BOTTOM LINE

Creating the ***visual design*** involves finalizing the type of graphics for the game and the respective design components based on the graphics type. Additionally, it involves deciding the UI layout based on the UI concept.

Visual design is the expression of the thought or idea of the game through graphical representation. It is a way of communicating the concept of the game to the target audience and urging them to buy the product.

Visual design plays an important role in targeting, engaging, entertaining, and motivating the audience. A good visual design ensures that your product looks different from the competition. Along with interesting gameplay, good visual design can make your game receive positive reviews from gamers. *Hitman, Final Fantasy, Counter Strike*, and *Fable* are all examples of games with good visual design.

The primary steps to creating the visual design for your game are:

- Selecting the type of graphics—2D or 3D
- Creating the design components
- Selecting the UI of the game

Selecting the Graphics Type

*The **graphics type*** is the medium of graphics that the game designer uses to create the game design elements.

The first step to creating your game's visual design is to decide the type of the graphics you are going to use in your game. Based on this, you can choose the appropriate design elements and the suitable game engine for your game application.

GRAPHICS TYPES

There are two main types of graphics—2D and 3D.

2D graphics are a blend of images and/or text created in a two-dimensional medium. A game created with 2D graphics shows only the height and width but not the depth of the game objects. Old cartoon movies and early video games used 2D graphics.

2D graphics require very little processing and so they do not require heavy graphics processing unit (GPU) support. That is why most mobile games—and some PC games—primarily use 2D graphics. Examples of such games include *Angry Birds, Snake*, and *Solitaire*.

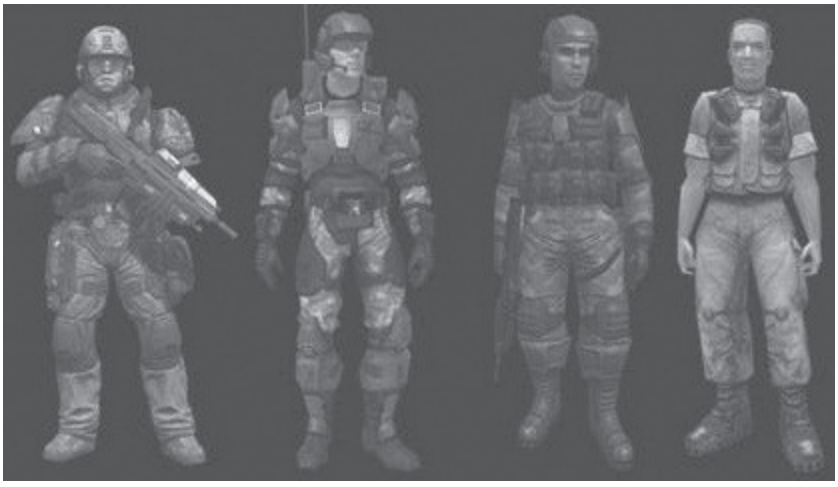
In contrast to 2D, 3D graphics are created in three dimensions: height, width, and depth. All real-world objects are three-dimensional. Therefore, you can use the 3D graphics medium in your game to provide the player with a real-world gaming experience.

www.itbookshub.com

The 3D medium allows you to create a character or environment for the game with different levels of detail. The more detail you add, the more real the character or environment will look. Almost all platforms with good GPU support 3D graphics. Games such as *Halo, Counter Strike*, and *Final Fantasy* use 3D graphics. Figure 3-1 shows the 3D images of game characters in the *Halo* game with low, medium, and high details from left to right, respectively.

Figure 3-1

3D game characters from *Halo* with varying details



©Microsoft Corporation

TAKE NOTE*

Today, in the video game industry, 2.5D is the most popular and commonly used graphics technique. 2.5D is a hybrid between 2D and 3D, in which images fake three-dimensional objects, however the actual medium is two-dimensional.

Consider the following factors while choosing the 2D or 3D graphics type for your game:

- Target audience
- Game output device
- Game platform

To better understand how these factors influence the choice of the graphics type, let us consider two different cases.

- **Case 1:** Assume that your target audiences are people in the age group of 16 to 22 years of age. Also, assume that your audience use smartphones as the game platform with a screen display of 960 by 640 pixels. In this case, the graphics of the game should be such that the visibility is not hampered by too many user interface elements occupying too much screen space. Also, the user interface elements should be large enough for the player to interact with the game through the small size display screen. In addition, the person's mobile device might or might not be apt to handle the latest 3D graphics. Considering these factors, in general, 2D graphics type becomes a choice of preference for the given game platform. However, it is not a mandatory choice.
- **Case 2:** Assume that your target audience consists of people who are avid travelers. Assume that the target platform is a tablet, which has a fairly larger display than a smartphone. For that reason, the graphic designer of the game has the liberty to introduce comparatively more user interface elements in the game. The players also have enough screen space to focus on. Even the user interface elements can be comparatively smaller in proportion to the ones for a smartphone. In this case, 3D graphics may be the better choice.

CERTIFICATION READY

Which graphics type is better for creating a game for mobile devices—2D or 3D?

3.3

Creating the Visual Design Elements

Visual design elements create the visual design of your game characters and game objects.

You can incorporate the artwork in your games through images. These images—2D or 3D with the required visual design—are an important factor in influencing the player's choice of a game. The different design elements that help to create and enhance the visuals of your game graphics include:

- Bitmaps
- Vector graphics
- Sprites
- Text
- Sprite font
- Textures
- Lighting
- Blending
- 3D geometry
- Parallax mapping

BITMAPS

A ***bitmap*** is a file format that is used to store images. A bitmap image—also known as a raster image—is made up of pixels or dots that contain rows and columns of little squares. Each pixel is a different color. You can edit the individual pixels of a bitmap image. The file size of a bitmap image is large because the file stores information about individual pixels. When you resize a bitmap image, you can view individual pixels of the image as the image zooms in. This makes the image lose its quality. Figure 3-2 shows a bitmap image and its scaled up version.

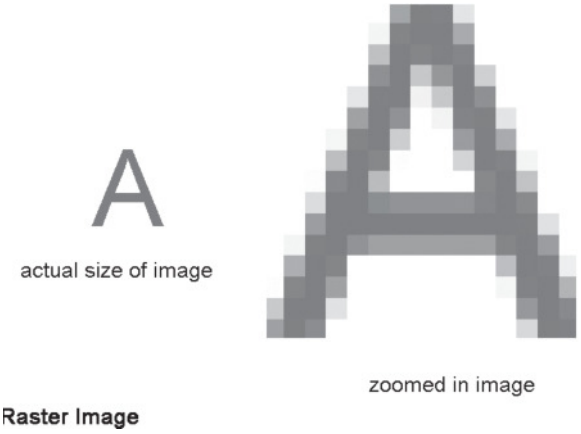


Figure 3-2
Resizing a bitmap image

Bitmaps are either device-dependent or device-independent. Device-dependent bitmaps have a .DDB file extension and, as the name suggests, are stored in a format specific to the device. That is, device-dependent bitmaps are stored in a format close to the form in which the device driver displays the bitmap. For this reason, device-dependent bitmaps can render faster on the display device.

In contrast, device-independent bitmaps, which have a .DIB or .BMP file extension, are stored in a format that any device can interpret. However, these bitmaps are slower in rendering on the display device.

Bitmap files are available in both compressed and uncompressed formats. Some of the file formats that support compressed bitmaps are GIF, PNG, TIFF, and JPEG. Uncompressed bitmap file format include BMP.

You can use bitmaps to represent images in your game, but this file format uses more memory to store images. Because of this, you should use the JPEG or DirectDraw Surface (DDS) graphics file format to represent images in your game.

Figure 3-3 shows a bitmap used to create a blood splatter that provides an indication to the player that a game character is dead.

TAKE NOTE*

The DDS file format was developed by Microsoft and is generally used in 3D games. This format saves memory on the video card.

Figure 3-3

A blood splatter bitmap

VECTOR GRAPHICS

Unlike bitmaps that use pixels, vector graphics use geometrical shapes—such as points, lines, and curves defined by mathematical calculations—to create images. A vector image contains many objects. Each object has individual properties, such as color, shape, thickness, and fill. Vector graphics store information about the individual objects in a structure. This helps in providing smaller file sizes for images stored in vector formats. Some of the commonly used vector file formats include Scalable Vector Graphics (SVG), Windows Metafile (WMF), and Adobe Illustrator (AI).

One primary benefit of vector images is that they are scalable. This implies that whenever you zoom in on a vector image, the clarity or quality of the image is not lost. In addition, vector images are based on mathematical calculations; therefore, you can manipulate the individual elements of the picture. However, this can hinder the screen performance. There can be a delay in the display because repetitive calculations are done whenever an image or frame is moved.

TAKE NOTE*

Flash supports vector graphics. In online games, the client machine or the browser does the processing of the vector images.

In the early video games, use of vector graphics was rare, except in arcade games. However, in recent years, with the increase in the rendering power of the output devices of the different platforms, video game production teams commonly use vector graphics for their images. Also, gamers use different output devices with varying screen resolutions. Therefore, because the vector images can scale, developers can provide pleasing results whenever the images are

displayed in different screen resolutions. Figure 3-4 shows a vector image and its scaled up version.

Figure 3-4

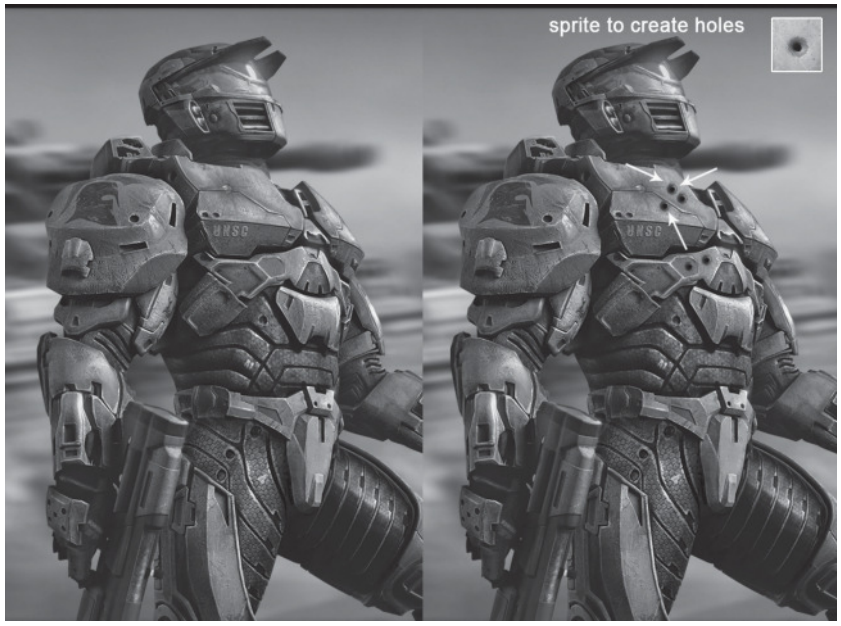
Resizing a vector image

SPRITES

A sprite is a two-dimensional plane on which an image is applied. The image can be anything—a leaf, stone, tree, mountain, or even a character. When included in a larger scene, these objects appear to be part of the scene. In video games, you can use sprites to combine small unrelated bitmaps into a single bitmap on the screen. For example, consider an action game in which the hero shoots the villain in the chest using a gun. In a real-world scenario, the bullet creates a hole in the person's chest. To mimic this in the game, you can create an image of a chest with a hole in it by using a sprite (see Figure 3-5). You can then connect the sprite to the villain's chest to give the player a feel of the real-world experience.

Figure 3-5

A sprite mimicking bullet holes in the chest



©Microsoft Corporation

You can use another sprite to show blood on the character's body (see Figure 3-6). The encircled part in the image shows the sprite effect for blood.

Sprites help keep the game light. To understand how, consider an adventure game in which you create a dense forest. A dense forest can be created by combining some close-up tree models (3D)

Figure 3-6

An example of a blood sprite

©Microsoft Corporation

with sprites (2D pictures of trees). The rendering of a sprite is faster compared to the rendering of all the facets of a 3D model at a cost of detail. If the tree (or other object in the sprite) is small enough on the screen, the loss of detail does not result in a loss of realism.

A sprite is one of the most important design components that help in the smooth flow of the game. However, sprites are two-dimensional. Therefore, when they are used in a three-dimensional virtual world, they appear to float without blending into the scene, especially when the player views the scene from an angle. To overcome this limitation, the sprite's rotation is connected to the movement of the camera or the player in the required scene. Wherever the camera or the player moves, the sprite tends to rotate to the camera's angle or the player's angle, so it blends in with the 3D virtual world.

TAKE NOTE* Many games use billboarding to make sprites (2D objects) appear to be 3D. Billboard is a technique in which 2D objects are applied to a rectangular primitive or a polygon that is kept perpendicular to the line of sight of the player or the camera.

TEXT

Early computer games used text characters because computers did not have graphics capabilities. *Adventure,* also called *Colossal Cave Adventure,* and *Genocide* are examples of text-based video games, which depend on written words or text characters. They have little or no graphics. Some text-based games, such as multi-user dungeons (MUDs), are multiplayer online games. In these games, gamers can read or view textual descriptions of the objects or characters of the game. They can also read descriptions about other gamers and actions carried out in the virtual world. The gamers interact with each other and with the virtual world by typing text commands.

Text-based games have the bandwidth to handle any number of users at a time. Their hardware requirements are basic compared to graphics-based video games. Therefore, players can play text-based video games using any computer connected to the Internet. Moreover, even if a player has a slower network connection, it does not hinder the gameplay.

Despite these advantages, today, with the advancement of computers and gaming technology, text-based games are becoming obsolete. The use of text in games has reduced to providing instructions, game-level information, messages, warnings, and other important information to the player. Most engines allow Windows-based fonts to draw text elements in games.

SPRITE FONT

In general, fonts are displayed on computer systems using vector graphics. This implies that fonts are defined by mathematical calculations. However, when you use fonts in your game, it increases the processing time because enormous time is spent in computation cycles whenever each font is displayed. To avoid this, the XNA Framework provides you with sprite fonts. The sprite fonts work by converting a normal vector-based font into a bitmapped font. The bitmapped font can be rendered quickly without involving complex computations.

Figure 3-7 shows a car racing game. The game uses sprite fonts to display the position of the cars, the time, and speed of the respective speeding car at run time.

Figure 3-7

A car racing game displaying sprite fonts

Sprite fonts can also be used to display the winning or the losing status of the player at run time. Figure 3-8 states that the player has not won the game by displaying the text, "You Lose."

Figure 3-8

A sprite font indicating losing status

TEXTURES

TAKE NOTE*

You should be careful to avoid copyright issues while using fonts in your game. Microsoft and other online sites provide free fonts.

Texture is a 2D image that is applied to a game object or character to define its surface. You can use textures to create the look of the object. For example, suppose your game has a wall object. To show a brick wall, you can create the required texture and apply it to the wall. In another scenario, let us assume a character in your game depicts an office worker. You can use texture and apply it to the plain three-dimensional image of the character to create the look and feel of a real office worker. Figure 3-9 shows the three-dimensional image of an office worker character with plain surface. Figure 3-10 shows the same office worker character with texture applied to the 3D surface.

Figure 3-9

A three dimensional image with plain surface

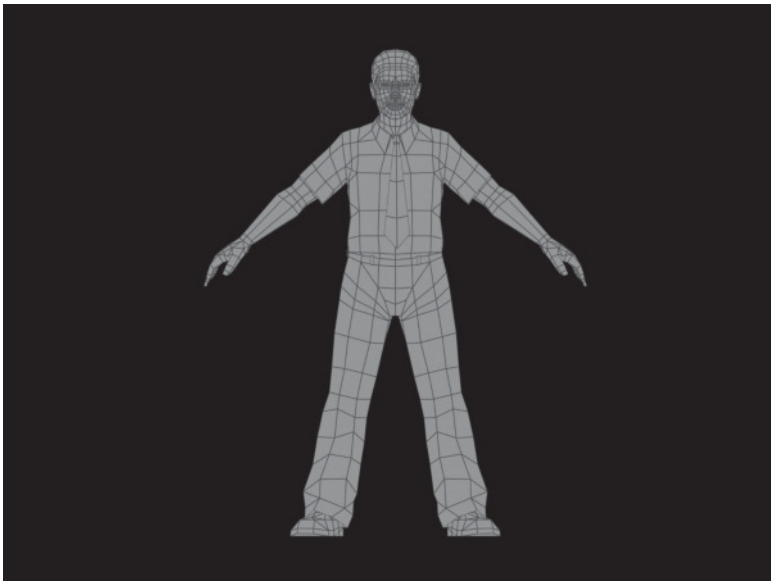


Figure 3-10

A three dimensional image with texture

Textures help to enhance not only characters, but game objects as well. Figure 3-11 and Figure 3-12 show the textures that can be used to enhance stone objects and tree objects in a game.

Figure 3-11

The stone texture

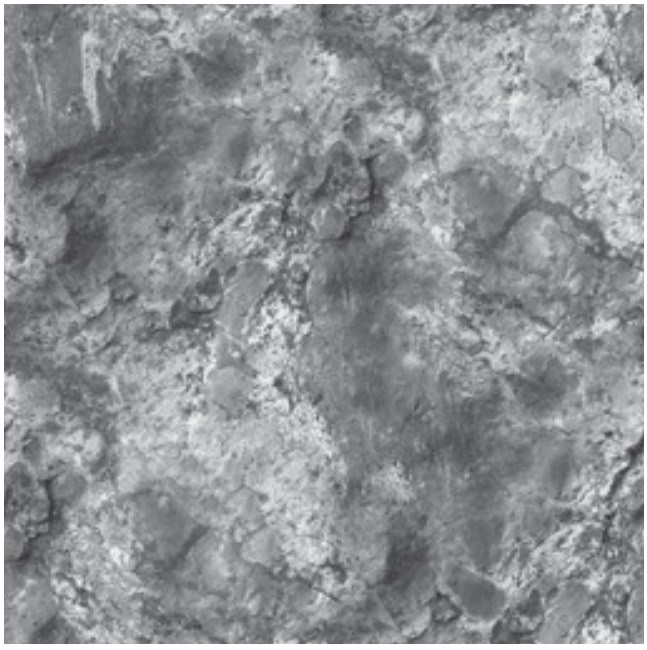


Figure 3-12

The tree texture

LIGHTING

Lighting helps to enhance the aesthetics of your game. By designing, focusing, and plotting the lighting component of your game, you can enhance the game visuals and make your game objects resemble real-world objects.

Figure 3-13

A screen shot from *Halo: Reach* showing the use of lighting

©Microsoft Corporation

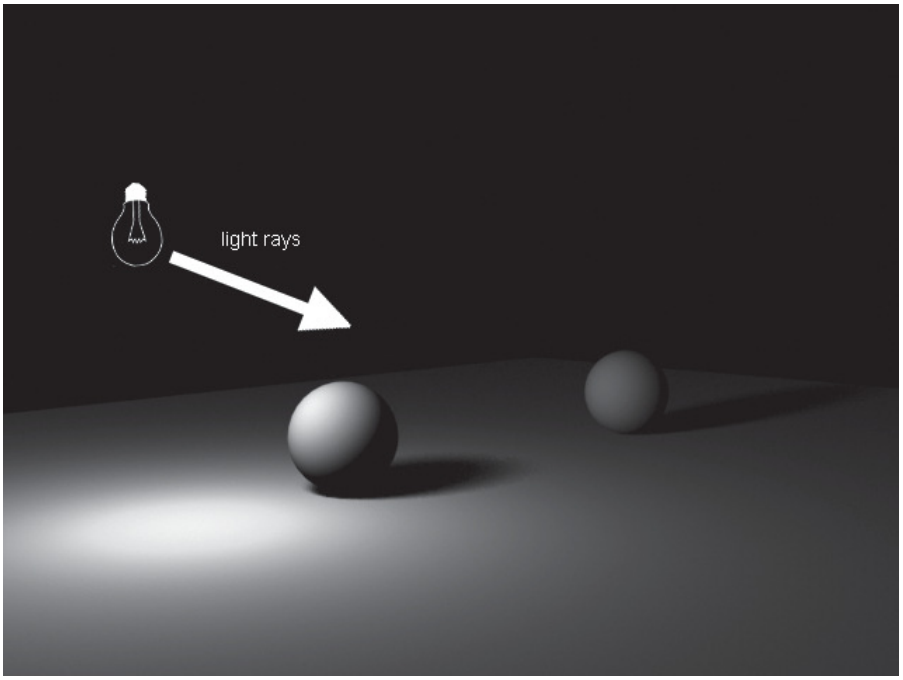
You can use lighting to set the time and the scenario of the game. Figure 3-13 shows a screen shot from the game *Halo: Reach*. In the given screen shot, you can see that the shadow of the character falls on the ground indicating that the scene is taking place in a broad daylight.

You can supplement the 3D feel of your game objects by providing light effects with proper composition. That is, objects near the light source should have more lighting and objects farther away from the light source should have reduced lighting. For example, Figure 3-14 shows the composition of light on different balls. The ball near the light source has more lighting and, as the light travels farther, the lighting diminishes and the second ball appears to have less lighting. In this figure, there is grading in light. On the other hand, in Figure 3-15, the composition of light on the balls does not have any diminishing effect. Both of the balls look bright no matter how far they are from the light source.

At this point, think of how a movie captures the lighting effect that exists in reality. However, because games involve imaginary animated characters or objects and not real-life characters or

Figure 3-14

Effect of light with decay



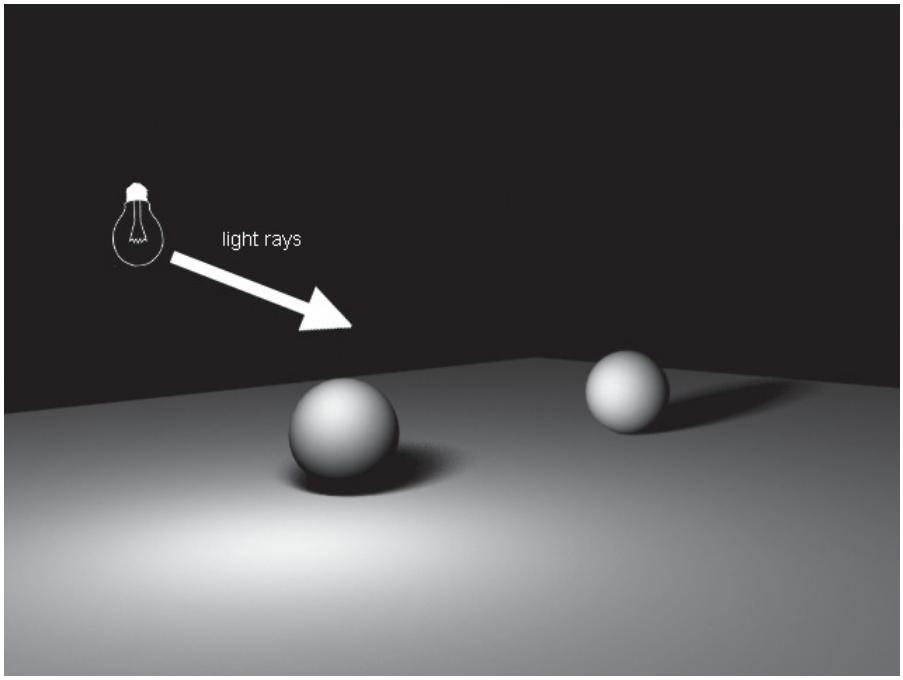


Figure 3-15
Effect of light without decay

objects as in movies, providing proper lighting in games becomes a difficult task. It involves calculating the resulting color of the surrounding objects or characters when light falls on them. The level of detail will, in turn, increase the size and complexity of the game. For example, suppose a character in a game stands below a light. Depending on the complexity of the game, the game might or might not be programmed for moving the shadow of the character on the floor with respect to the position of the light. This calculation requires complex computations that the game engine and the GPU might not be able to handle.

To make this task simpler, nowadays game engines provide predefined properties and functions to create commonly used light in computer graphics. Examples include light point, spot, directional, and ambient light. Certain game engines also provide lighting systems that can mimic the natural day and night lighting.

BLENDING

Blending is the mixing of a new color with an already existing color to create a different resulting color. Blending helps to create many special effects in your game visuals. Each color that is mixed contains a blend factor. The blend factor defines how much of each color is mixed in the final color. You can specify the blend factor for each of the colors and use the respective blend function to get the required effect.

Microsoft XNA Game Studio 4.0 provides predefined properties and functions that you can use for blending. Different blend modes are available (see Figure 3-16).

Figure 3-16 illustrates four common blend modes, namely alpha blending, additive blending, multiplicative blending, and 2X multiplicative blending from left to right, respectively. The images at the top row in the figure are source images and the images at the bottom row show the effect of blending when added to the destination.

The most commonly used blend mode is additive alpha blending. Before discussing additive alpha blending, let us define alpha blending. Alpha blending is the process of combining a foreground translucent color with a background color to create a transparency effect in order that the destination color appears through the source color. In computer graphics, alpha means transparent or semi-transparent pixels. Additive alpha bending is a process in which the source and the destination color is added to get the resultant color. For example, if your source color is

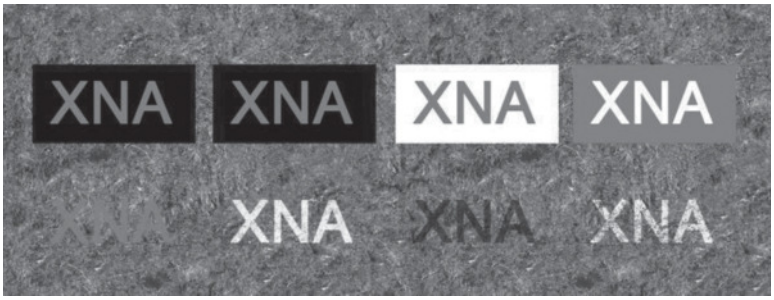


Figure 3-16

Examples of different blending modes in XNA

blue and the destination color is red, the resultant color that you get is purple. You can use additive alpha blending to create 3D atmospheric and environmental effects. For example, in your action game, suppose there is a scene of a bomb blast in a house. The house has red lighting. You use a sprite element for the smoke that arises due to the blast. To give a realistic effect and to make the sprite element part of the scene, you use additive alpha blending. After blending the smoke, which is white in color, with the red light source in the room, the smoke will look red.

MORE INFORMATION

You can refer to the section "What Is Color Blending" in the MSDN Library to learn more about the blend settings in XNA 4.0 for different blend modes.

3D GEOMETRY

A 3D geometry is an object that has three dimensions: length, width, and height. In computer graphics, you can use 3D graphics to represent 3D objects. To enhance the gamer's experience, most video games today use 3D graphics.

3D graphics need a lot of GPU processing. Therefore, heavy graphics might make your game run slowly, or might even shut it down. Therefore, while using this design element in your game, you should consider the following points:

- Make sure that your game model adapts to different graphics levels and output devices. A game contains different graphics levels. In general, a game engine is designed in such a way that it can iterate a model depending on the setting the player chooses and also depending on the distance of the game objects from the camera. That is, if the object is far away from the camera, the game engine loads the low quality data of the object and if the object is near the camera, it loads the high quality data of the object. Figure 3-17 shows a game model from low to high graphics level.
- Your game graphics become heavier once you apply texture to it. Therefore, keep the base geometry simple and light.

As already discussed, assume your game has a character that depicts the image of an office worker. To portray this image, you can apply appropriate texture to the base geometry of the

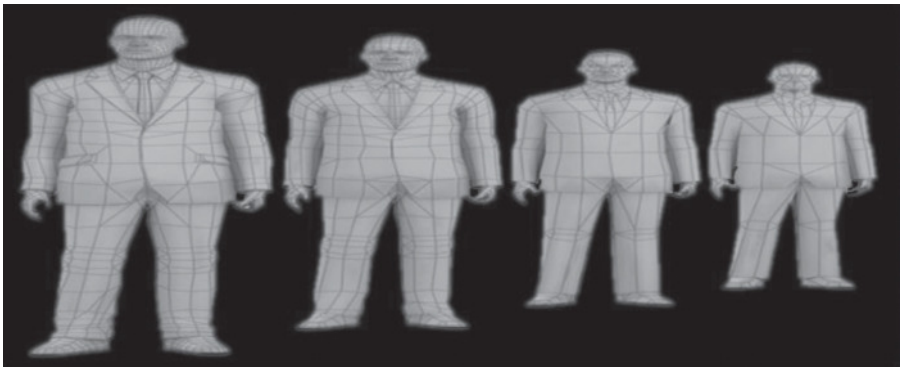


Figure 3-17

A game model in different graphic levels

model. You can revisit Figure 3-9, which shows the plain 3D model of a game character, and Figure 3-10, which shows the 3D model with the texture applied.

PARALLAX MAPPING

In video games, ***parallax mapping*** is a 3D technique that is an enhancement of the "normal mapping" technique applied to textures. To understand parallax mapping better, first let us define normal mapping. In computer graphics, normal mapping is a technique to fake the lighting of bumps and dents on game objects and characters. This technique helps to give depth to a game object by applying a normal map on the game object. A normal map is generated from a high polygon model. It is an RGB (red, green, and blue) image, in which the RGB components corresponds to the X, Y, and Z coordinates of the surface normal, which is a vector that is perpendicular to the surface at that spot. Parallax mapping is an enhancement of normal mapping. Using parallax mapping techniques, you can make a game object, such as a stone wall, look like a three-dimensional object, with more apparent depth and with minimal effect on the performance of the simulation. Parallax mapping displaces the individual pixel height of a surface, making the high coordinates hide the low coordinates behind them. This gives the illusion of depth on the surface when looked at an angle.

You can use parallax mapping to enhance the look of the flat textures painted on a 2D surface. For example, consider the rock shown in Figure 3-18. Although the rock has the required texture, it looks unrealistic. By applying parallax mapping to this image, you can add depth to the rock and make it look real (see Figure 3-19).

CONSIDERATIONS FOR GOOD VISUAL DESIGN

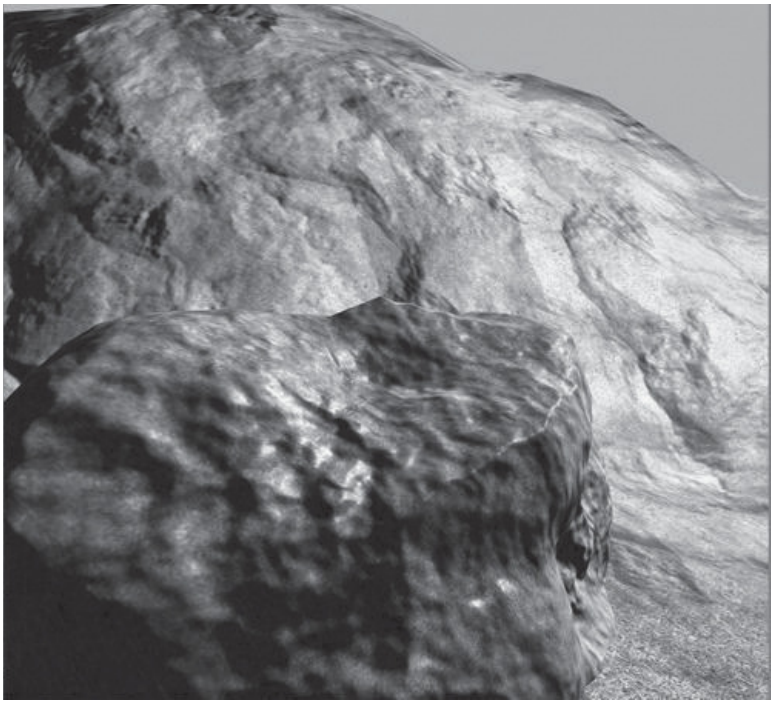
All the game design components covered so far can help you design a game that looks more realistic and engrosses its audience. However, that will happen if you create the game design components considering these points:

- **Simplicity:** You should always keep the visual design of your game simple and clear so that players can easily use and follow the game play.
- **Compatibility:** The visual design of a game's characters or objects should be compatible with the game theme and game concept. For example, consider a war scene in an action

Figure 3-18

A rock without parallax mapping

Figure 3-19
A rock with parallax mapping



game. The visual design of the characters in this game should depict real-world soldiers, with bloodstains and mud on their clothing. The characters should not look neat and tidy.

- **Clarity:** Every character in a game should be clearly distinguished from other characters with respect to physical features. For instance, let us take the example of an action game in which a hero fights with the villain to save a victim. In this case, if the victim's visual physical features and the hero's visual physical features are not distinguished properly and do not depict the respective character type, then the gamer might get confused and lose track of who is who.
- **Use of colors:** Use proper colors for the characters and objects. For example, suppose the game has a high-action scene, such as a big team fight. If you use simple colors and no value contrast in the game, then all the characters in the fight will blend into each other. This can make the players lose track of the characters.

CERTIFICATION READY
What are all the design elements that you can use for an FPS game?
3.3

Selecting the UI Concept and Layout

UI layout is the arrangement of the UI elements in the game. The UI layout for a game is designed based on the ***UI concept,*** which should be in line with the game concept.

UI layout constitutes all the UI elements, including the interactive elements and the noninteractive elements. Interactive UI elements include buttons, text fields, and menus even the game characters through which the audience interacts with the game. The noninteractive elements include game objects such as trees, forests, and islands, which provide the environment for the game. Both interactive and noninteractive elements are designed based on a UI concept that in turn is derived from the game concept and the visual theme conceptualized for the game.

To ensure that you create a game that the player can easily play, you must select the UI layout and concepts carefully. A thorough understanding of the UI concept will help you select the UI elements that best suit your gameplay. For example, in the *Angry Bird* game, the UI can retain the game concept and have a visual theme consisting of birds, eggs, nests, and mazes. The UI elements can be designed to reflect this UI concept.

Your game UI acts as an interface that binds the player's input with the actions in the game. The UI also helps the player to get messages conveying the details of acts or occurrences of events from the game. For an engaging gaming experience, the UI of your game must be consistent, easy to use, informational, and attractive.

A game with a good UI makes the player enjoy the game. On the other hand, a game with a poor interface frustrates the player even though it has a good game concept. Therefore, creating the visual design for your game is one of the most challenging activities in creating the UI of the game. It is challenging because, one, you need to convey all the necessary information to the player within the little screen space available, and two, you need to do that effectively and without cluttering the screen.

The interactive and noninteractive UI elements of a game can be categorized into four types of ***UI components***. These UI component types might be part of the game story or game space, or completely fall outside the game space or game story.

The UI components that form part of the game narration or game story, act as an interface between the game avatar and the player. The game avatar and other game characters are aware of these UI components.

The UI components that form part of the game space provide an interface between the game world and the player. Players have to use the UI components to participate in the game world. The game avatar and game characters are not aware of the UI elements.

Let us now look at the four UI component types in detail.

UI COMPONENT TYPES

The four primary UI components include diegetic components, nondiegetic components, spatial components, and meta components. Let us look at each of these components in detail.

Diegetic Components

Diegetic components can exist within the game story and the game space. They assist the player by providing indication and information about the game world, in which the game avatar and other game characters are aware. A map is an example of a diegetic component that the player can use to indicate where he or she is located (see Figure 3-20).

You can use diegetic components to enhance the narrative experience of your game, and also the player's involvement in the game. For example, in *Counter Strike*, the ticking sound of the bomb

Figure 3-20

A map indicating the current location

and the timer are diegetic components. In this game, the terrorist plants the bomb in one place. The player character can hear the ticking sound of the bomb from a far corner and, as the character approaches the place, the sound becomes louder. When the bomb is about to explode, the character can hear the ticking sound faster, and the bomb eventually explodes. The timer component in the game indicates to the player character the time remaining before the bomb explodes.

Nondiegetic Components

In contrast to the diegetic components, nondiegetic components are not part of the game story or the game space. You can use these components to enable the player to choose the game setting. For example, you can use nondiegetic components to allow players to customize the type of weapons they want to use in the game (see Figure 3-21).

These components can have their own visual treatment and players can completely customize them. Figure 3-22 shows an opening screen of a sample game that uses a menu as a nondiegetic component. Note that the title, "Ultimate Carnage," displayed on the screen is a fictional name and not the title of any real video game.

Figure 3-21

Customizing the weapon using nondiegetic components

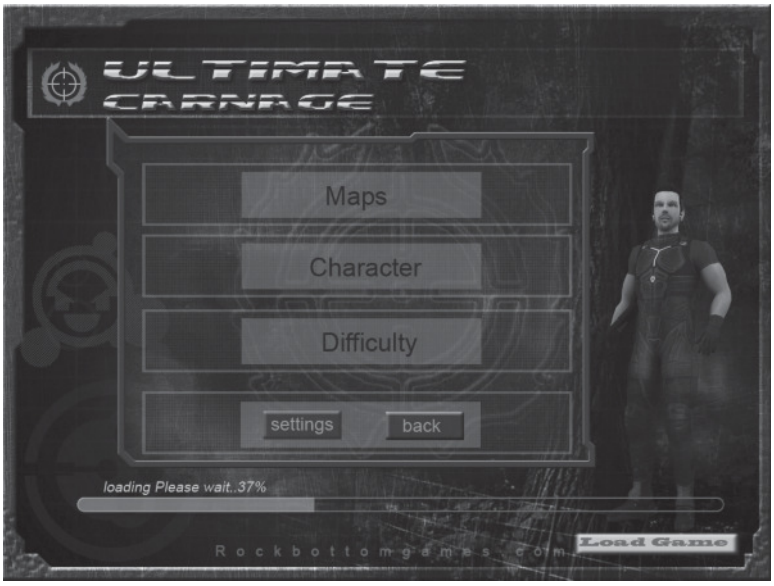


Figure 3-22

Use of nondiegetic components in a sample game

Spatial Components

Spatial components are part of the game space but not part of the game story. They provide extra information on a game object or character to the player, eliminating the need for the player to jump to menu screens to seek more information. In this way, this component maintains the gaming experience of the players. For example, you can use spatial components in a game to tell the player which weapons they can gather on their way to a mission (see Figure 3-23).

Figure 3-23

Use of a spatial component to indicate weapons in store

Meta Components

A meta component exists as part of the game story alone. Meta components are usually used with diegetic components to recreate a real-world experience. However, meta components are difficult to define in a game that does not have a strong storyline. You can use meta components to express effects such as a blood spatter or cracked glass. These effects convey messages to the players in order to draw them into the reality of the virtual world. For example, the onscreen blood spatter in a game conveys that the player is hit (see Figure 3-24).

Figure 3-24

Use of meta component for blood splatter

Players expect games that are simpler to use and easier to play. You must therefore strike a balance between immersion and usability. To create a game that is engaging and also provides the optimal gaming experience, you must select the UI layout after careful consideration of the pros and cons of each UI type. Table 3-1 presents a quick comparison of the different UI components.

The decision to choose the right components based on the two methods is quite tricky. Some game designers argue that using components that form part of both, the game story and the game space (Purpose 1 and Purpose 2), helps in creating games that are more engaging. Other game designers counter-argue that these components at most provide a visual appeal, but present only limited information to the player. Often, the information presented through these components causes the player to misinterpret data and make errors during the game. This reduces the player's gaming experience. For example, in *Dead Sea*, the holographic 3D map was not interactive enough to navigate the player. The map did not act like a real-time GPS system; rather, it was just a noninteractive element for making the game attractive. A nondiegetic component could have easily replaced it.

Table 3-1

Comparison of UI components

UI Component	Pros	Cons
Diegetic	• Enables the player to connect with the game world. • Helps in connecting the storyline with the game.	• Seems contrived or forced if the UI elements are not represented properly. • May not necessarily provide proper information to the player. • Not suited for games in which there must be a break in the game to provide critical information to the player.
Nondiegetic	• Enables UI elements to have their own visual treatment. • Helps in overcoming limitations imposed by other UI components.	• Does not immerse the player into the gameplay as the diegetic components do.
Spatial	• Helps to separate information to the player from information to the player's character. • Player does not need to tab between screens to take in information.	• Can seem forced if the elements are not required.
Meta	• Ensures replication of real-world experience through blends with diegetic layout. • Presents clear information to the player.	• Can create confusion and can easily distract the player from the actual game play. • Player may waste game time in searching for this kind of layout. This might lead to distraction from the actual gameplay. • Requires a good storyline for the use of layout.

Having discussed the primary types of UI components, let us now look at some UI elements that represent these UI components.

UI ELEMENTS

The UI elements that can represent the four types of UI components are menu, button, HUD, text, text field, icons, and many more interactive UI elements. You can use these UI elements to represent any of the four UI components based on your game requirements. Let us discuss some of the UI elements in brief.

Menu

You can use menus in your game to provide the player with a list of options. The player can choose from the list as desired. You can use menus as part of your game story or as part of your game space. You can also place a menu as a nondiegetic component on a welcome or opening screen, where players can select the activity they want to perform. For example, you can use menus on the opening screen of your game to allow the player to choose a previously saved game or start a new game. Alternatively, you can use menus in the game story to provide options for the player character to select a particular game setting (see Figure 3-25). For example, in a war game, you can use menus to allow the player character to select a weapon.

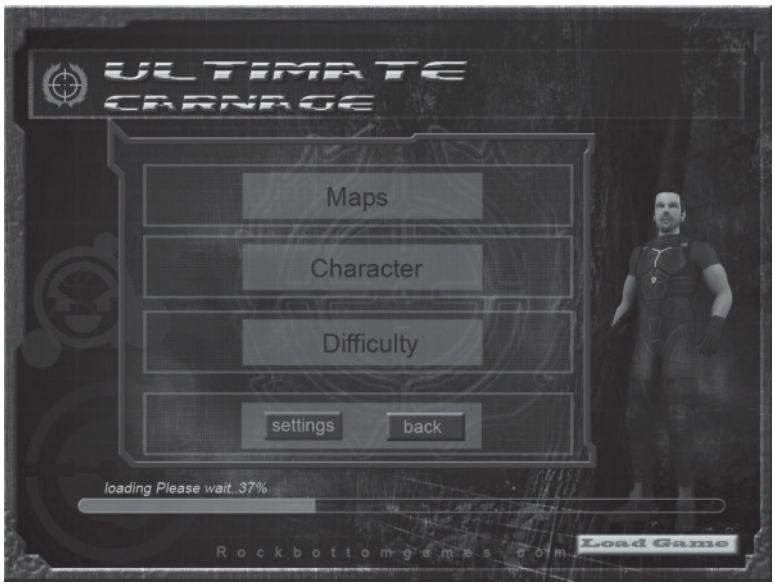


Figure 3-25
A sample menu

Follow these guidelines while using menus in your game:

- Keep the menu navigation fast by keeping the UI menu code light. Players do not like to wait for menus to load or, worse, to wait for the selected option to load in the middle of a game. So use only basic coding for the menu. Menus with light code appear faster on the screen and cause minimal interruption for the player.
- Keep the UI menus well organized. Players who use PCs select a menu item precisely with a mouse pointer. However, players who play console games use the D-pad or thumbstick in the game controller to select the required menu item by passing through the entire menu. Therefore, you should design your menu aptly, keeping the platform requirements in mind.

- Keep the menu scrolling minimal. Scrolling menus make the players select and evaluate every choice until they find the right one. However, if you keep the menu scrolling minimal with all the options visible, players can directly find their choice (see Figure 3-26).

Figure 3-26

A scrolling menu

Heads-Up Display

A heads-up display (HUD) UI provides game-specific information to the player through visual representations. The HUD can help you take the game story forward. For example, a key HUD can help you open a door that can take you to the next level of the game.

Generally, HUDs convey information using numbers or text, but many other visual representations can be used as per the game type and UI concept. Some of the common information types that you can display using HUDs and their representations include:

- **Character's health/life status:** You can simply present the health status through a health bar (see Figure 3-27). For example, in a shooting game, you can provide a health bar that drains out to indicate that the character has just suffered a hit.

Figure 3-27

Examples of HUDs in a shooter game

- **Weapons:** You can show the weapon that the character is using and the balance amount of ammunition that the character has, as shown in Figure 3-27, or the choice of other weapons that are available. You must consider the player's need when selecting the information in relation to the weapons. For example, you can display the choice of weapons and the current weapon being used so that the players can change weapons frequently during the game, if they want to.
- **Menus:** Menu UIs can also be used as HUDs. However, you should carefully select the menu items that you want to include as HUD. For example, menus to change the settings, exit the game, or pause the game can be included in the HUD, but certain other menus, such as saving the game, accessing the help menu, and so on do not need to be part of the HUD because these activities will rarely be performed during the game.
- **Game-specific visual elements:** You can use certain visual elements, such as a speedometer or a compass, as HUDs to help the players achieve their objective. For example, you can use a speedometer to show the speed in racing games or a compass to indicate the direction in quest games. You can indicate the range of the target and the ammunition status of the weapon in shooting games (see Figure 3-27).
- **Time:** You can use HUDs to indicate the time elapsed during the game, or the balance time in a time-bound game, or even the current time of day.
- **Game status**: You can display the current game level, score, and tokens using HUDs.

Consider the following while using HUDs in your game:

- Provide information using HUDs that will best motivate the player to continue playing the game.
- Decide whether the information displayed through the HUD remains on the screen at all times, or whether the player can access the information when required through some keys depending on the gameplay requirement. You can also choose to allow the player to hide a part of the HUD.
- Keep your HUD transparent so that the players do not need to change their viewpoint to view the information.
- Keep only the most relevant information in the HUD. Avoid overloading information, which might confuse and frustrate players. Include only the most relevant information in the HUD.

The HUD is an important visual element of the game and provides scope for creative designs. For example, the *Dead Space* HUD moves along with the player's character, but does not obstruct the gameplay. All vital information, such as air, health, map, inventory, and so on, is displayed through the HUD. *Assasin's Creed* is another game with a good HUD. The highlight of this HUD is the use of corners to place important information, such as health, map, available actions, and weapons. The HUD does not distract the player but can still be easily viewed.

Buttons

You can use buttons to allow the player to perform specified actions when the player clicks the buttons. For example, you can provide a Start button in the opening screen of your game, so that the player clicks the button to begin playing.

The following is a list of considerations while using buttons:

- Keep consistency in form and design of the buttons across the game.
- Keep the button design so that it clearly stands out from the rest of the visual elements. Players are more inclined to click on buttons than on text.
- Use fonts that provide a smooth display and are easy to read even in small font sizes.
- Use filters such as Drop Shadow, Glow, and so on, but only if it is an absolute necessity.

CERTIFICATION READY
What is a suitable UI concept for a sports game in the mobile platform?
1.4

Congesting your game's UI with too many options can confuse the player, resulting in a frustrating gaming experience. Therefore, keep your game's UI layout simple. The UI of your game should contain only necessary UI elements. Your game players should be able to navigate the game with ease.

■ Deciding the Output Parameters

THE BOTTOM LINE

The outputs of a game that a gamer finally views are not only influenced by the type of input/output devices, but also depend on different factors.

The factors that affect the game output include:

- The medium used to render or deliver the visual output or the graphics of the game
- The different resolutions at which the game might run
- The techniques used to compress the video and audio output

CERTIFICATION READY
What is DirectX?
3.1

You already know that graphics cards enhance the graphical output of a game. Different graphics cards have different capabilities in displaying graphical information. Your game application communicates with the graphics card through the respective device drivers. Certain graphics application programming interfaces (APIs), such as DirectX and OpenGL, provide abstraction to this communication. A ***rendering engine***, which is a renderer for 2D or 3D graphics, is built on graphics APIs.

Rendering Engine

Rendering is the process of generating a 2D image from a 3D object model using rendering engines.

The model that depicts a game object or game character is defined by mathematical representation of points and surfaces and is created by a game designer or a game artist. You already know that game engines provide a software framework that helps in the rapid development of games. Moreover, game engines also include a rendering engine. A rendering engine abstracts the communication between the graphics hardware, such as the graphics-processing unit (GPU) or video card, and the respective device drivers through a set of APIs.

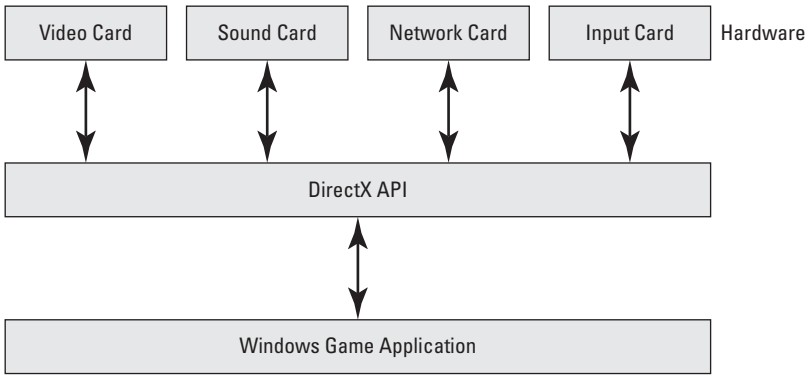
Some game engines available in the market exclusively provide real-time 3D rendering capabilities. Such game engines are often termed *3D rendering engines* or *3D engines*. The task of 3D rendering engines is to convert 3D models to a 2D image with real-time 3D effects. Examples of 3D rendering engines include Crystal Space, OGRE, Truevision3D, and Vision Engine. One of the commonly used 3D rendering engines is Microsoft's XNA Game Engine. However, the XNA Game Engine wraps around the functionality of the DirectX Software.

DirectX

The ***DirectX*** application is a collection of APIs from Microsoft that handles high performance multimedia applications including games. DirectX contains a set of dynamic-link libraries (DLLs), which you can use to develop hardware-independent game code to access different computer hardware components, such as input devices, network cards, and sound cards. If the game player's system configuration does not have the necessary hardware that a

game might require, DirectX software emulates the functionality of the missing hardware. For example, say your game requires the video card to rotate a bitmap. If the video card (hardware) of a player's system does not support rotation, then the DirectX emulates the functionality through software algorithms. Thus, software emulation occurs whenever hardware lacks the required feature or functionality. Figure 3-28 shows the interaction between the hardware, DirectX, and the Windows-based systems.

Figure 3-28
Interaction between DirectX and hardware devices



From Figure 3-28, you can infer that DirectX provides your game application with greater accessibility to hardware devices such as:

- Video cards
- Input devices, such as joysticks, keyboards, mice, and joypads
- Sound cards
- Network cards

In addition, DirectX also controls 2D acceleration chips. It controls sound mixing and sound output on different audio hardware. With every new version of DirectX, more and more new features are added to help game developers develop games for the latest software and hardware technologies, such as the latest operating system versions, the latest graphics cards, sound cards, network cards, and other drivers available on the market.

TAKE NOTE*
Hardware acceleration is the process of using separate computer hardware to enhance the speed of computing. Therefore, because the GPU helps the CPU by calculating graphics computation, it is sometimes also referred to as a graphics accelerator.

CERTIFICATION READY
What are the different APIs included in DirectX and for what purposes?
3.1

The collection of APIs in DirectX includes Direct3D, Direct2D, DirectSound3D, DirectPlay, DirectInput, and so forth. The "X" in DirectX refers to the respective API names. Let us look into some of the DirectX APIs in brief.

- **Direct3D:** Direct 3D is a set of 3D graphics API within DirectX that you can use to develop games for Windows operating systems (95 and above) and Xbox 360 console. Direct3D uses the advanced features of 3D graphics cards. You can use Direct3D to enhance the entire graphical feel of the game by providing better display. Combined with other DirectX technologies, Direct3D helps to deliver 3D rendering in 2D planes and sprites.
- **Direct2D:** Direct2D contains the 2D graphics API that provides high performance and high quality rendering of 2D images. It interoperates well with Graphics Device Interface (GDI)/Graphics Device Interface+ (GDI+) APIs, and Direct 3D APIs. It improves

the performance of your game by using the hardware acceleration through compatible graphics cards. Direct2D also helps you to display your game in different dots per inch (DPI) settings by allowing automatic GUI scaling. DPI is a measure of resolution, in particular the number of discrete dots that can appear in a line within the span of one inch. DPI measurement depends on various display resolutions and screen sizes.

- **DirectSound:** DirectSound is a set of APIs that provides communication between multimedia applications, including your game applications and the sound card driver. DirectSound helps in rendering sounds and the play back of music. Additionally, DirectSound also provides recording and sound mixing of multiple audio streams and adds effects such as echo and reverb to sound. The latest version of DirectSound can play multichannel surround sound at high resolution. DirectSound3D is an extension of DirectSound that places the sound using a three-dimensional coordinate system instead of a simple left and right pan. DirectSound3D enhances the realistic effect of sound in a 3D virtual world. Today, DirectSound and DirectSound3D have been merged and commonly referred to as DirectSound.
- **DirectPlay:** DirectPlay provides a set of APIs that provides an interface between your game applications and communication services, such as the Internet or local networks. In simple terms, DirectPlay forms a layer above network protocols such as TCP/IP. It allows the players to interact with each other regardless of the underlying online service or network protocol. It helps you to provide your game players with game sessions and manages the flow of information between hosts and players.
- **DirectInput:** DirectInput is a set of APIs that helps your game application collect input from the players through the input devices. The input devices can be of any type, such as a mouse, keyboard, other game controllers, and even a force feedback. Moreover, this API helps you to assign game-specific actions to the buttons and axes of the respective input devices. DirectInput uses action mapping, which allows the game to retrieve input data without worrying about the source input device. Action mapping allows you to associate each action in your game with one or more of the built-in DirectInput virtual controls. The DirectInput then takes care of mapping these actions with the choice of the available input devices in a system. In this way, it allows the player to select and configure the input devices of their choice. When your game runs, DirectInput matches the input received from the chosen devices and maps them to the virtual controls that are associated with the respective game specific actions.

TAKE NOTE*

A force feedback device, also called a haptic feedback device, works in conjunction with onscreen actions within a game. There are many types of force feedback devices such as game pads, joysticks, steering wheels, and so on. These devices give the players feedback in the form of vibrations when they are shooting a gun or hit by an enemy. For example, when a player is shooting a machine gun in an action game, a force-feedback joystick device vibrates in the player's hands, creating realistic forces.

DirectX 11 is the latest version of DirectX. DirectX 11 has improved features that help you provide stunning visuals in your game and also improve game performance significantly. It helps you provide improved computing and hi-speed, highly reliable gaming.

TAKE NOTE*

Microsoft has not revised DirectInput API since DirectX 8. Microsoft has introduced XInput in place of DirectInput for Xbox 360 controllers.

Understanding Display Initialization

Display initialization is a set of minimum requirements that should be available for your game to run smoothly on the chosen platform.

CERTIFICATION READY
What are the display initialization requirements for a mobile game in the educational genre?
3.1

Every game has some bare minimum requirements without which the game will fail to initialize on a given platform. You can specify the kind of GPU that is required for the game, and you can specify the components that are necessary for your game to function properly. For example, if you use DirectX as the rendering engine and your game has music, your game will not function properly if the DirectX sound component is not present in the player's system. In this case, your game might display an error message or it might not run at all, creating great frustration for the user. To avoid such situations, you must list the necessary setup requirements that the user needs to check before initializing the game.

SETUP REQUIREMENTS

Before discussing the requirements you need to specify to your game user, let us look at an example of an error message that players might get while initializing their games.

Sample error message: A player gets the error message "could not initialize display hardware" when he tries to run the game on his machine.

Reasons: This message might appear due to the following reasons:

- **Reason 1:** The game could not start because the display hardware or the GPU is not capable enough to run the game. The video card memory requirement is not enough or the make of the video card is not compatible with the game. For example, video cards such as the E.G Intel graphics cards are not suitable for games.
- **Reason 2:** There might be some limitation in the player's display hardware.
- **Reason 3:** There might be another application or process running on the player's machine, which might hinder the initialization of the game.
- **Reason 4:** The files that are installed during game installation might be corrupt.

Now let us look at some of the requirements that you can specify for your game. This helps the user to provide the specified requirements on the platform before playing the game.

- Specify the display hardware, that is, the kind of GPU that can help your game graphics run smoothly.
- Specify the type of application that needs to be shut down for the game to initialize properly. For example, for *Counter Strike*, if the machine on which the game runs also has a WIBU-KEY Server Application running, then the performance of *Counter Strike* on the network will slow down. In this case, it is necessary to inform the user to turn off the applications.
- When gamers install your game on a PC, the files installed on the machine might become corrupt due to some virus present on the machine. To avoid this, you can specify that users should check for viruses before they install your game on their machines.
- You can inform users about any diagnostic tool that they can use to check the capabilities of the hardware installed on their machines. For example, DirectX Diagnostic Tool helps users troubleshoot problems that are encountered while playing games.

The following list contains the minimum display initialization requirements for the *Halo: Combat Evolved Anniversary* game released in November 2011:

- Microsoft Windows 98, Microsoft Windows Second Edition, Microsoft Windows Millennium Edition (Me), Microsoft Windows 2000, or Microsoft Windows XP
- 733 megahertz (MHz) processor
- DirectX® 9.0 or later
- 128 megabytes (MB) of RAM
- 1.2 gigabytes (GB) of free hard disk space
- 32 MB with 3D Transform and Lighting capable
- CD: 8X

- Sound card, speakers, or headphones with multiplayer play
- *56.6 Kilobytes per second (KBps) modem or 10 MB network adapter

Understanding Resolution

Resolution is the number of pixels that can be displayed on a display device, such as a monitor or a television. The output quality of a game is good or bad depending on the resolution and size of the display device.

The word "resolution" is often quoted as "display resolution." You can express the resolution of a display device in terms of the number of pixels on both the horizontal and vertical axis. The resolution is cited as "width × height." For example, if we read out the resolution as "1024 × 768," it means that the width is 1024 pixels and the height is 768 pixels.

The quality of the image displayed on a display device depends on the resolution and size of the device. The resolution can be high or low. In a lower screen resolution, fewer items fit on the screen. However, the images look larger. In a higher resolution, more items fit on the screen. In this case, the images look smaller. Therefore, the quality of the image displayed on a small size screen will be great when compared to the quality of the image displayed on a larger screen. This is because the image tends to pixelate or spread out over a larger number of inches. Therefore, you should consider the proposed screen resolutions for your game in the beginning of your game design.

Games can be played in varying screen resolutions. Let us now see some examples of games that are played in different resolutions. We can know how resolution affects the design and performance of the game through the following examples.

Example 1:

The video card is GeForce 9500 GT 1GB DDR2.

The monitor is a 19-inch LCD with a native resolution of 1280 × 1024 pixels.

The games are *Grand Theft Auto IV* and *Call of Duty Black Ops*.

The games run smoothly on the display device with the prescribed resolution. A change in the resolution also does not affect the performance of the previous games.

Example 2:

The video card, monitor, and the resolution remain the same for another game: *Call of Duty World at War*. However, a change in the resolution to 1024 × 768 affects the game with respect to its performance.

From the previous examples, we infer that the design of the game depends on the video card and in turn on the resolution.

If you create your game explicitly for a particular platform, you can design the game accordingly, aiming at the resolution of the screen of that platform. However, again, if the specific platform supports varying resolutions, it can cause a problem. For example, suppose you develop a game for the mobile platform. The platform can be a variety of devices, such as a mobile phone, tablet, and so on, with varying screen sizes. In such cases, you need to create different graphics for every resolution, which is not the optimal choice. An alternative is to set up a fixed video resolution and tell the end user to adjust the monitor to the resolution.

Another factor to consider while deciding on the resolutions is the ***display mode***. Games can be played in different display modes on a display device. These display modes have different resolutions. However, game engines today are smart enough to adapt to the resolutions of the

respective display mode. You can set the required mode for your game using the settings provided in the game engine.

DISPLAY MODES

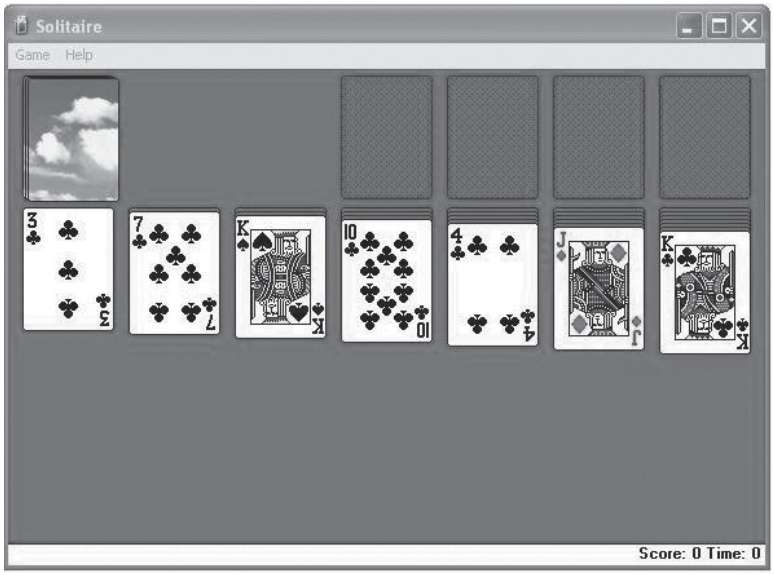
CERTIFICATION READY
What are the possible display modes that you can select for an MMORPG game?
3.1

The display modes include full screen and Windowed.

In the full screen mode, the game is displayed on the full screen. This mode does not display the title bar of the game. An example of a game that can be played in full screen mode is *Pac-Man*. The advantage of running this game in full screen mode is that the gamer cannot see the menu or the button displayed by the operating system. This helps the player to immerse in the game without distraction. Games run on a console or mobile can be viewed only in full screen mode. However, games that are run on PCs can display in other display modes as well. When you develop your game in full screen mode, with certain settings your game can adapt to different kinds of GPU, which adapts the game to different screen resolutions.

In the Windowed mode, the game is displayed in a single window on the screen. Certain games, such as *Solitaire*, can be played only in a window kind of interface. Other games, such as *Maze* and *Pac-Man*, which do not require much GPU support, can be run on this mode. Only PC users can use the Windowed mode to play games. This mode helps the user perform other tasks as well on their computer. However, this increases the load on the PC, as it has to handle multiple tasks. Figure 3-29 shows the game *Solitaire* in the Windowed display mode.

Figure 3-29
Solitaire in Windowed mode



One common problem in both the display modes is that irrespective of the resolution on which your game might run, your game visual might look distorted because of the higher frame rates produced by video cards. This can be overcome by using ***Vertical Synchronization (VSync)***.

VERTICAL SYNCHRONIZATION

CERTIFICATION READY
What is the role of VSync in a game display?
3.1

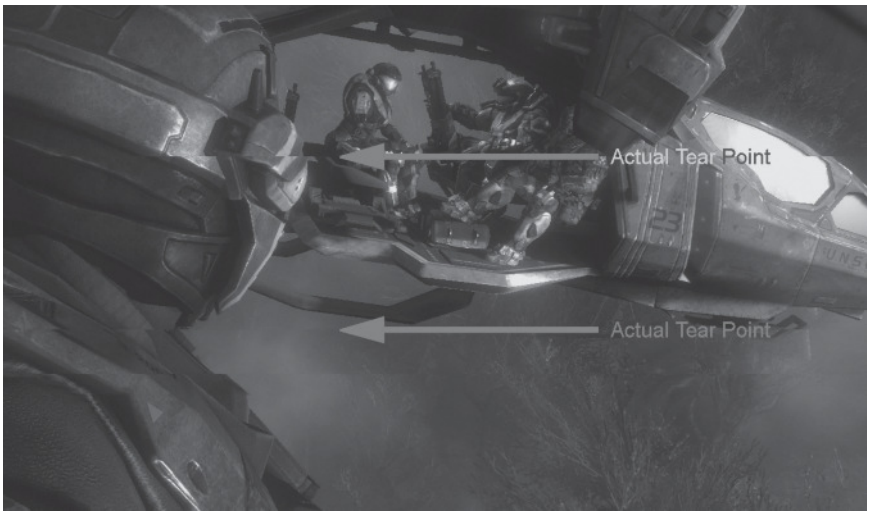
Before understanding VSync or Vertical Synchronization, let us understand how a monitor displays an image. In general, the computer draws the image to be displayed on the monitor repeatedly. The speed of redrawing is referred to as *frame rate*. Whenever the computer refreshes the displayed image, whatever we see on the monitor changes. However, if the computer refreshes this image even before the monitor has completed drawing it, it might look as if the image is torn. VSync provides the solution to this problem.

VSync controls the refresh rates of your display. It restricts the video card from producing frame rates higher than the refresh rate of the monitor. In other words, VSync makes the computer wait until the monitor has drawn the image completely. Therefore, the number of times the image is drawn cannot be greater than the monitor refresh rate. The common refresh rate on displays is 60 HZ or 60 frames per second (fps). This reduces the load on the graphics card because VSync sets a limitation on the number of times the image is drawn.

You can turn VSync on or off. If VSync is turned off, the frame rate exceeds the native refresh rate of display. This causes "tearing" of the image. Tearing creates a torn look because edges of objects displayed in the visual fail to line up (see Figure 3-30). The frame breaks into two or three parts. Sometimes, the current frame falls behind the other part of the same frame. These kinds of visuals bother game players.

Figure 3-30

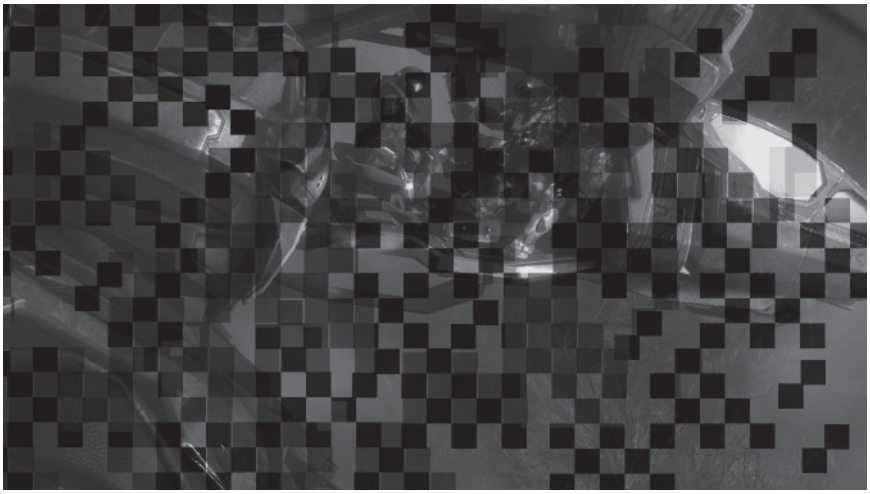
An example of the tearing effect



The solution to avoid tearing is to turn VSync on. However, in this case, the VSync locks the refresh rate of the monitor with the frame rate in multiples of 60 for a refresh rate of 60 fps. It causes the video card to render images only when the monitor is finished drawing. In such cases, if there is a drop in the frame rate, then VSync will also drop accordingly and cause a lag in the image display, which is call *stuttering* (see Figure 3-31). This makes the game speed up and slow down alternately, causing great inconvenience to gamers.

Figure 3-31

An example of stuttering



TAKE NOTE*

Adaptive VSync is part of the Release 300 drivers and is compatible with GTX 680 and GeForce video cards.

Adaptive VSync fixes both tearing and stuttering problems. It dynamically changes VSync on and off. When Adaptive VSync is turned on, VSync also turns on, thus eliminating tearing. Whenever the frame rate drops, VSync shuts off automatically, thus eliminating stuttering. Adaptive VSync also reduces the power consumption of the video card. To turn on the Adaptive VSync settings, you need to go to the control panel. The new Adaptive VSync mode increases the playability of the game. It enriches the video card to make the game faster, smoother, and richer in terms of experience.

Understanding Audio and Video Compression

Audio and video compression is the reduction of the audio/video data to fewer bits using respective compression techniques.

CERTIFICATION READY
What are the different audio and video compression techniques?
3.1

Games use large video and audio files, which occupy extensive storage space on a computer system. Additionally, network games increase the load on the network and take a longer time to download or upload. The best solution to this problem is to compress the data in these files. ***Compression*** is the reduction of the data to fewer bits by following certain techniques. The advantage of compressing an audio or a video file is that the compressed file occupies lesser space and, in addition, is transferable quickly through the available communication channel. A compressed file can have no or very negligible modification in its quality with reference to the original file.

You need to possess the required skills to compress audio and video files in a game. You need thorough knowledge of the available compression techniques, the various compression factors that need to be considered, and the suitable methods to perform compression of a specific game without modifying the quality of its image or sound.

UNDERSTANDING THE BENEFITS OF COMPRESSING GAMES

During the initial years of gaming, games such as PS2, Gamecube, and Xbox had to compromise on the quality of the audio. This was because the better quality audio files occupied too much space. It was also because the mixing and tuning of the sounds for a particular session in a game was brought about by decreasing or eliminating unnecessary sounds with the application of real-time interactive mixers and digital signal processing techniques.

The video files, graphics, and animations in a game occupy even more space than the audio files. However, advancements in compression techniques of audio and video files have provided a solution to this problem.

Today, compressing a game file implies decreasing the size of the audio and video data files. The main advantages include increase in the speed of the game and reduction in the storage space on the disk.

ENABLING COMPRESSION

Data redundancy is the main factor that enables compression. In general, there are four types of redundancies in audio or video files. Table 3-2 lists these four types and their descriptions.

CATEGORIZING COMPRESSION TECHNIQUES

We can classify the compression techniques into two main types, lossless and lossy.

Lossless compression, as the name suggests, accounts for data reduction without any data loss after compression. Therefore, the original data can be reconstructed from the compressed data without losing any information. It is also known as bit-preserving or reversible compression system.

Table 3-2

Data Redundancy Types to Facilitate Compression

Data Redundancy Type	Feature
Spatial	Within an audio or a video frame, there exists considerable correlation between the adjacent samples or data items.
Spectral	When collecting data from multiple sources (for example, cameras), there exists considerable correlation between the samples from these sources. This method uses the frequency area and focuses on the relationship between the occurrences of changes in the data.
Temporal	There exists considerable correlation between the samples in different time segments. This is applicable for 1D (one-dimensional) data, 1D (one-dimensional) signals, and audio.
Psycho-visual	Makes use of the perception signals from the human visual system.

Lossy compression, on the other hand, involves some loss of data during reduction. The original information cannot be reconstructed when the file is decompressed. This technique is useful for compressing audio and video files. The main advantage of this technique is that it neither compromises the quality of the compressed file with a given bit-rate nor reduces the bit-rate to achieve the desirable quality.

ANALYZING THE COMPRESSION PROCESS

Every compressed audio or video file is associated with a CODEC, which stands for *coder-decoder*. A ***codec*** is a program that facilitates lossless or lossy compression and decompression of digital files.

A video codec facilitates compression or decompression of digital video files by mostly applying the lossy data compression technique. Some codecs also use the lossless data compression technique.

Examples of video codecs include:

- MPEG-4 Part 2 codecs (DivX Pro Codec, Xvid, FFmpeg MPEG-4, and 3ivx)
- H.264/MPEG-4 AVC codecs (x264, Nero Digital, QuickTime H.264, and DivX Pro Codec)
- Microsoft codecs (WMV (Windows Media Video) and MS MPEG-4v3)
- On2 codecs (VP6, VP6-E, VP6-S, VP7, and VP8)
- Additional codecs (DNxH, Sorenson 3, Sorenson Spark, RealVideo, Cinepak, and Indeo)

An audio codec facilitates compression or decompression of digital audio data, with reference to a particular audio file format or streaming media audio format. Most codecs function as libraries with interfaces to one or many multimedia players. Examples of common audio codecs are MPEG 4 (Motion Picture Experts Group), WAV (Waveform), and MDI.

VIDEO COMPRESSION TYPES

There are several types of video compression techniques:

- **M-JPEG (Motion-Joint Photographic Experts Group) compression:** Involves intraframe coding for eliminating the spatial redundant data.
- **H.261 and MPEG compressions:** Involves intraframe and interframe coding for eliminating the spatial and temporal redundant data.

- **Avi (Audio Video Interleaved) compression:** Uses different compressors to balance the file size for the desirable quality. The main drawback is that the file size is large even after compression, which poses problems when transferring.
- **DivX compression:** Provides high quality compression. Supports multiple languages and provides good visual quality and fast performance.

TAKE NOTE*

Interframe coding uses successive frames in a sequence to compress the current frame in a video. In other words, this coding technique compares the differences and similarities of the current frame with the next frame in a video. It then processes only the significant changes in particular pixels. This coding provides enormous data reduction. Intraframe, on the other hand, compresses each frame in a video independently.

Let us understand some of these video compression types in detail, which are popularly used in games.

- **M-JPEG Compression:** M-JPEG involves intraframe coding only. The absence of interframe coding restricts its performance when compared with its counterparts, such as MPEG-1, MPEG-4, and H.264. M-JPEG has reduced processing and memory requirements.

  Frames with large, smooth transitions or monotone surfaces compress well, and keep their original detail with little visible compression of data. Each frame in the video is saved as a JPEG file.
- **H.261 Compression:** H.261 compression is well adapted for video telecommunication applications, such as video conferencing and video telephone applications over ISDN telephonic connections.
- **MPEG Compression:** MPEG stands for *Moving Picture Expert Group*. Many standards, such as MPEG-1, MPEG-2, MPEG-3, MPEG-4, MPEG-7, and MPEG-21, have been in use from time to time. Of these, MPEG-1 and MPEG-2 are finalized standards and are used currently in a variety of applications.

  MPEG-1 focuses on video resolutions from 352 × 240 pixels at 30 fps (for NTSC video system) and 352 × 288 pixels at 25 fps (for PAL video system) up to 4095 × 4095 pixels at 60 fps. Its common name is Source Input Format (SIF) video system and it is suitable for only progressive scans. Therefore, it does not support interlaced video applications, such as in broadcast television applications. It is used for VCDs. MPEG-2 supports both progressive and interlaced scans. It is used for DVDs.

TAKE NOTE*

Interlaced video is a technique used for drawing a video image on a display device such as a television. This technique displays each line of pixels by using two fields for a single frame. One field displays all the odd lines in the image and the other field displays all the even lines of the image.

Progressive scanning is a technique in which all the lines in an image are displayed in a sequence.

AUDIO COMPRESSION TYPES

Some of the commonly used lossy compression types for audio data compression include:

- **Silence compression:** The silence compression method involves receiving speech samples, storing them in the memory, and analyzing the samples later to determine the silence periods. These periods of silence are then compressed and restored back in the memory.

- **Adaptive Differential Pulse Code Modulation (ADPCM):** This method involves conversion of analog sound data into a string of digital binary code. Regular samples of the analog sound are taken, and the difference between the actual values of each sample and its derived value (gathered from previous samples) is processed and converted to digital signal.
- **Linear Predictive Coding (LPC):** Linear predictive coding encodes audio signal at a low bit rate. This method separates the effects of sound source such as the vocal cords, and filter, the vocal tract from an audio signal. It encodes the data in a speech signal into a smaller space for transmission over a limited channel.
- **Code Excited Linear Predictor (CELP):** This method follows the process of the LPC method and transmits the parameters of the models at the rate 4.8 kbits/sec along with errors.

STREAMING AUDIO AND VIDEO

Currently, streaming audio and video types are the most common data delivery media for the Internet and other multimedia networks. Examples of streaming audio and video used in gaming solutions are RealAudio and Shockwave, respectively.

SELECTING DESIRABLE AUDIO FORMATS

A variety of file formats is now available for recording and playing digital sound and music files. Some depend on the software that needs to be installed for their application, whereas others need specific operating systems.

Before checking the available audio file formats, it is important to understand the difference between an audio file format and an audio codec. As discussed previously, an audio codec is a computer program that enables compression or decompression of a digital audio data. The process of compression or decompression is performed based on the specifications in a given audio file format. An audio file format is the format of storing the digital audio data (uncompressed or compressed) on a computer system.

AUDIO FORMAT TYPES

Audio format files are classified based on whether they store uncompressed or compressed data. On this basis, there are three types of audio format files:

- Uncompressed (Example: WAV audio file)
- Lossless compressed (Example: WMA audio file)
- Lossy compressed (Example: MP3 and RealAudio files)

Let us look at the different audio format files in brief.

- **WAV Audio Format File:** WAV audio format files have the .wav extension and store sounds in files developed together by Microsoft and IBC. Most of the applications in Windows support this file format.
- **WMA Audio Format File:** WMA audio format files have the .wma extension and compress data at a higher rate when compared with the MP3 file format. In addition, WMA can compress files of any size and makes them suitable for different connection speeds or bandwidths.
- **MP3 Audio Format File:** MP3 audio format files have the .mp3 extension and are derived from the MPEG file type with audio layer 3 for compressing the audio signals. For example, using this audio format results in compressing sound data with a bit rate of 1411.2 kilobits/sec to 112–128 kilobits/sec without compromising the sound quality.
- **Real Audio Format File:** Real audio format files have the .ra or .ram or .rm extension and help in playing digital audio files in real time. To use this file format, it is mandatory to install RealPlayer (for Windows or Mac), which is freely available on the Internet.

VIDEO FORMAT TYPES

A number of video formats are available. Some of the most popular ones are the following:

- **DVD Video Format:** DVD uses the optical storage technology. It can be recordable, such as DVD-R, DVD-RAM, and DVD-RW, or application-based, such as DVD-Video, DVD-Video Recording (DVD-VR), and DVD-DVD Audio Recording (DVD-AR). There are also customized DVD formats for game consoles, such as Sony PlayStation and Microsoft Xbox.
- **Flash Video Format:** The current versions of Flash, namely versions 6 and 7, support full motion video and streaming video, respectively. Flash is user-friendly and adapts to all environments. Users can create their own interactive media content with graphics and animations, and personalized media players with custom controls.
- **QuickTime Video Format:** QuickTime is a multimedia technology that efficiently manages video, audio, animation, music, and virtual reality environments.
- **RealMedia Video Format:** RealMedia is a multimedia technology and is widely used for streaming content over the Internet.
- **Windows Media Video Format:** Windows Media is one of the most popular technologies available for streaming or downloading audio or video.

TOOLS FOR COMPRESSION OF GAMES

Compression of games requires a variety of tools. Of these tools, the most common are Zencoder and Rad Game Tools.

TAKE NOTE*

The type of compression techniques for a game depends on the game engine that you use.

- **Zencoder:** Zencoder is an API-based video encoding service that is available online. It converts videos for a website, an application, or a video library into formats that are adaptable to the mobile phone or other desirable device. It supports mostly all the video and audio codecs. Zencoder mainly focuses on video encoding along with audio encoding.
- **Bink Video:** Bink Video is developed by RAD Game Tools. It is a video codec for games.

SKILL SUMMARY

IN THIS LESSON, YOU LEARNED:

- Graphics type means the medium of graphics that a game designer uses to create the game design elements.
- There are two types of graphics, 2D and 3D.
- 2D graphics depict images in the two-dimensional model.
- 3D graphics depict images in a real-time three-dimensional model.
- A graphical design element enhances the look and feel of images.
- The various visual design elements are bitmaps, sprites, vector graphics, lighting, blending, text, textures, 3D geometry, parallax mapping, and sprite font.
- The user interface (UI) layout constitutes all of the UI elements.
- UI concept is the idea behind the making of UI layout.
- UI components generally reside within the game story or within the game space. Diegetic, spatial, meta, and nondiegetic are the various types of UI components.
- Diegetic components exist within both the game story and the game space.
- Spatial components exist in the game space. They provide extra information on a game object or character to the player, which eliminates the necessity of the player jumping to menu screens to seek more information.
- Metacomponents exist as part of the game story alone. You can use metacomponents to express effects such as a blood spatter or cracked glass.
- Nondiegetic components are not part of the game story or the game space. These components can have their own visual treatment and players can completely customize them.

- Other UI elements that you can use in your games include the menu, heads-up display (HUD), and buttons.
- A HUD UI provides game-specific information to the player through visual representations.
- A rendering engine abstracts the communication between the graphics hardware, such as the graphics-processing unit (GPU) or the video card and the respective device drivers through a set of APIs.
- DirectX is the one of the commonly used rendering engines. It is from Microsoft and is a series of APIs that handles high performance multimedia applications including games.
- Some of the collections of APIs in DirectX include Direct3D, Direct2D, DirectSound3D, DirectPlay, and DirectInput.
- Direct3D APIs expose the advanced features of the 3D graphics card. You can use Direct3D to enhance the entire graphical feel of the game by providing better display.
- DirectSound is a set of APIs that provides communication between multimedia applications, including your game applications and the sound card driver.
- DirectPlay provides a set of APIs that provides an interface between your game applications and communication services such as the Internet or local networks.
- DirectInput is a set of APIs that helps your game application collect input from the players through the input devices.
- You have to provide your game user the set of minimum requirements that should be available for your game to initialize smoothly on the chosen platform.
- The word "resolution," which is often referred to as "display resolution," is the number of pixels that can be displayed on a display device such as a monitor or television.
- The design of your game depends on the video card, and in turn the resolution.
- The different display modes on which your game can be played are full screen, VSync, and Windowed.
- VSync helps your game run without tearing. However, it causes stuttering of the image displayed.
- Adaptive VSync is a new technology that eliminates the tearing and stuttering of images. This technology turns VSync on/off automatically as required.
- In the Windowed mode, your game appears in a window. Only PC users can avail this mode.
- Compression helps to condense the video and audio file size in a game.
- The two main types of compression techniques are lossless compression and lossy compression.
- CODEC (coder-decoder) is a program that facilitates lossless or lossy compression and decompression of digital files.
- Compression of games requires a variety of tools. The most common tools are Zencoder and Rad Game Tools.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ is a 3D technique that helps the game objects look three-dimensional, with more apparent depth.
2. _____ helps draw text in your game visuals.
3. _____ image is represented in pixels.
4. _____ is an example of a multiplayer online text game.
5. _____ API exposes the advanced features of the 3D graphics card.

6. _____ provides a set of APIs that helps your game application collect input from the players through the Xbox console.
7. _____ technology helps prevent tearing of the image by restricting the video card from producing frame rates higher than the refresh rate of the monitor.
8. _____ and _____ are the two types of compression techniques.
9. _____ compression provides a larger file size even after compression.
10. _____ compression type compresses the periods of silence in speech samples.

Multiple Choice

Circle the letter or letters that correspond to the best answer.

1. In your action game, you create a scene in which the player character assaults the villain character on his hand. Which of the following design elements can be used to create the effect of the broken fingers of the villain character?
   a. Sprite
   b. Parallax mapping
   c. Vector graphics
   d. Textures
2. You develop an adventure game. You are required to mimic a cell built with brick walls for a particular scene. Which of the following techniques would you choose to provide an illusion of depth on the surface of the wall?
   a. Sprite
   b. Parallax mapping
   c. Lighting
   d. 3D geometry
3. In your sports game, which of the following design elements would you choose to display the scores of the players?
   a. Sprite font
   b. Parallax mapping
   c. Sprite
   d. 3D geometry
4. You develop an action game. You want to provide options for the players to choose a suitable weapon before an attack. Which of the following UI components would you choose? (Choose all that apply.)
   a. Sprite
   b. Diegetic component
   c. Nondiegetic component
   d. Menu
5. In an action game, you decide to show the balance amount of ammunition in the player character's gun. Which of the following UI components would best display the information?
   a. HUD
   b. Nondiegetic component
   c. Spatial component
   d. 3D geometry
6. You would like to provide the users of your game with game sessions. Which of the following APIs do you think would facilitate game sessions?
   a. DirectInput
   b. DirectPlay
   c. DirectSound
   d. XInput

7. You are creating a game for PC users. Which of the following display modes would you choose for your game? (Choose all that apply.)
   a. Full screen
   b. Windowed
   c. VSync
   d. Adaptive VSync
8. You decide to use M-JPEG compression in your game. Which of the following data redundancy types does M-JPEG Compression eliminate?
   a. Spatial
   b. Spectral
   c. Temporal
   d. Psycho-visual
9. You use compression tools to compress your game. Which of the following are compression tools? (Choose all that apply.)
   a. Zencoder
   b. Bink Video
   c. DivX Compression
   d. Code Excited Linear Predictor (CELP)
10. In your game, you decide to choose video and audio formats that are available by default in all the computer systems. Which of the following formats should you choose?
    a. AVI and WAV
    b. MJPEG and WMA
    c. H.261 and Real Audio
    d. H.261 and WMA

■ Competency Assessment

Scenario 3-1: Selecting Graphics Type

You are creating a game for mobile devices. Explain with comparison among 2D and 3D which graphic type will best suit your game.

Scenario 3-2: Using DirectX

You develop a console game. Explain the different DirectX components your game might use.

■ Proficiency Assessment

Scenario 3-3: Developing Visual Theme and UI Concept

You are developing a fantasy game for the mobile platform. Develop the visual theme and the UI concept for your game.

Scenario 3-4: Selecting Rendering Engine and Design Components

You are developing a mobile game. Your game concept is similar to *Angry Birds*. Select the suitable rendering engine and decide on the required game design components for your game.

4 LESSON

Designing Specific Game Components

EXAM OBJECTIVE MATRIX

Skills/Concepts	MTA Exam Objective	MTA Exam Objective Number
Designing Game States and Loops	Plan for game state.	3.2
Designing Objects and Characters	Animate basic characters. Transform objects.	4.1 4.2
Designing Physics-Based Animations	Work with collisions.	4.3

KEY TERMS

anisotropic filtering
collision detection
collision response
filter
fixed step game loop
frames per second (fps)
game loops
gameflow
general-purpose computing on graphics processing unit (GPGPU)
graphics pipeline
High Level Shading Language (HLSL)
interpolation
linear filtering
mipmap
nonplayer characters (NPC)
per-pixel lighting
physics engine
physics processing unit (PPU)
physics simulation
pixel shader
point filtering
projection matrix
projection space
scene hierarchy
scripted events
shader
sprite animation
texels
texture mapping
variable step game loop
vertex shader
view matrix
view space
world matrix
world space

Steve Watson works at Contoso Gaming Inc., and leads the development team. His company is developing a first-person shooter (FPS) game for console and computer systems. The team has designed the game's visual world. They are now in the process of designing various components for their game using XNA 4.0.

As a first step, Steve and his team decide to provide a good gameflow for their game. For this, they identify a well-designed sequence of challenges and rewards for their game in order to move the game story forward. They sequence the challenges in such a way that it increases the game's complexity at each level successively. They also decide to utilize every element of the game efficiently through optimizing the states of their game. This ensures that their game runs smoothly across different platforms. The central component of any game design is game loop. Steve's team implements the main loop in XNA to help the game run smoothly irrespective of the player's input.

Players interact with a game through objects and characters. To make these objects and characters come alive on the screen, Steve's team adds the minutest detail to these game elements, such as the types of movements characters can make, the tasks they can perform, and so on. Moreover, the team also decides to simulate artificial intelligence for their game characters to help retain the interest of players.

■ Designing Game States and Loops

THE BOTTOM LINE

You should design the gameflow, the properties of the objects and characters required at each game state in the gameflow, as well as the actions the characters can perform. It is also important to understand the scope of artificial intelligence (AI) in your game so that the end product lives up to the developer's and player's expectations.

The design of the game does not end with creating the visual design. You need to design the progress of the game from one game state to the next. You also need to decide which characters and objects will be present at each game state and the actions they will be able to perform. In addition, you need to decide whether your game or the characters in the game are going to have built-in intelligence, or AI.

The game components that you need to design at this stage include:

- Gameflow, game states, and game loops
- Objects and characters
- Physics-based animations and AI

Creating Gameflow

Gameflow is the progression of the game from one state to another state. It comprises a sequence of challenging tasks and provides rewards to players to motivate completion. Gameflow makes players experience the game and gives them a sense of accomplishment.

CERTIFICATION READY
What are the different aspects of a gameflow?
3.2

Video games often have a series of challenging tasks or complex situations. The goal of the game is for the players to complete the tasks and solve the situations to receive rewards. The reward can be anything—moving to a higher game level, another life, loot, new weapons, access to specific areas of the game, extra points, and so on. These rewards help motivate the

players to play the next level or take up the next challenge and continue in the game. Good gameflow ensures that although the tasks are challenging, the players can complete them. If the tasks cannot be completed even after multiple tries, the players can get frustrated and give up on the game.

The features of a gameflow are:

- Challenge
- Pace
- Scripted events
- Instinctive training areas
- Trial and error
- Instinctive prompt
- Player vocabulary

CHALLENGE

Challenges are the key source of interaction in a game. A challenge motivates the players to use their logic and skill to perform some tasks or solve a puzzle to move ahead in the game. Without challenges, the game becomes boring. For example, a simple game of *Tetris* can become boring if there is no challenge of preventing the blocks from reaching the top of the window. The challenge forces the players to think and use logic to place the blocks in the appropriate positions.

On the other hand, it is also necessary to remember that the challenge must have a solution. If the players do not complete the challenge even after multiple tries, they might stop playing that game. Thus, when designing a challenge, you must remember to make it easy to resolve after applying logical thinking.

You must also remember that you should not force the players into using a specific course of action. As a designer, you might think of a means of resolving the puzzle for completing the task. However, the players might not be able to relate to that logic and, as a result, would not be able to complete the challenge. Therefore, it is essential that the challenge has more than one solution; two or three solutions are ideal. This increases the chances of the players completing the task or solving the puzzle, boosts their confidence, and motivates them to take up more challenges.

PACE

Pace refers to the rate at which the excitement happens in a game. A game should aim to keep the excitement level of a player high until the end of the game.

The pace of a game might vary with the genre of the game. For example, the pace of an action game is faster than a logic or strategy-based game.

Another factor that affects the pace of a game is the skill level. A game should provide different options to different players depending on their skill levels. For example, in the minesweeper game, the mines to be identified increase in number with the increase in difficulty level from beginner to expert.

Alternatively, the player's skill can be gradually enhanced by providing learning opportunities at the initial levels of the game. This helps the players to be ready for more difficult challenges in the later levels.

Another option to maintain the pace of the game is to set time limits to complete a particular task or puzzle.

In short, a designer should not force the pace level on the players. Let the players play at their pace. A perfect solution to maintain the pace of the game is to provide different styles of playing the game at each level.

SCRIPTED EVENTS

A game can have ***scripted events***, which are minor objects or events that are programmed. These scripted events are usually triggered by an action from the player.

Let us use the example of a game that has an object with the image of fire on it. When the player clicks that object, fire is displayed on the screen.

The best use of scripted events is to teach the various aspects of a game to the players. These events do not interrupt the flow of the game but simply, provide an opportunity to the players to learn about a move or a feature of the game without leaving the game.

In some situations, you can even give scripted events an important role in the gameplay. For example, when a player reaches a certain area in the game world, you can design the game to provide the player with a special treasure or reward. You can design specific events to be triggered, such as the opening of a particular door as the player reaches it, or the falling of stones as the player reaches a stone bridge, and so on.

INSTINCTIVE TRAINING AREAS

All games have certain obstacles that the players have to overcome or puzzles that they need to solve to win the game. But if the players are unable to continue in the game by removing the obstacles, they might feel frustrated and abandon the game. One way to ensure that the players do not leave the game is to provide them certain areas on the screen where they can experiment their moves or tactics without affecting the progress of the game. Such areas are known as instinctive training areas. These areas allow the players unlimited attempts to perfect their moves and tactics. Players can then use the same moves and tactics to overcome the obstacle or solve the puzzle.

TRIAL AND ERROR

Trial and error is a method that facilitates learning. The players can make a move and view the consequences of the move. They can then modify the move to get better results. Trial and error gives the players an opportunity to learn about the consequences or results of different moves. The players can try various moves before finalizing a particular move. However, when designing trial and error, the designer must ensure that the consequences of an error do not lead to player frustration. For example, suppose there are four stages to a game level. The player has finished the first two stages. In the third stage, the player makes a wrong move. The result of this move is that the player is taken to the beginning of the level and has to play the first two stages again. If this process keeps repeating, the player might get frustrated and not try to move ahead in the game. Therefore, care should be taken that trial and error is a way of teaching the players and not punishing them. In addition, the designer should also ensure that there are limited incorrect moves. If the player plays too many incorrect moves, it might lead to player dissatisfaction.

The best solution is to use the trial and error method in the initial stages of the game so that it gives the player ample opportunities to become familiar with the game. The designer can then include extensions or variations to the moves used in the initial stages to increase the complexity of the game.

INSTINCTIVE PROMPT

Instinctive prompt is based on the player's learned behavior that he or she gains from computer knowledge. The player's learned instinctive moves are the correct moves that help the player to progress in the game. No clues, visual or instructional, are required for such moves. For example, when you use an application and you see a button, you click it without anyone telling you that it needs to be clicked. This is an instinctive move. You can incorporate instinctive moves in your game. For example, you can enable the players to use the up arrow to move forward and the down arrow to move backward. These are moves that players

are comfortable making and which give them pleasure because they are making these intelligent moves on their own without any help. This further motivates the players to proceed with the game.

PLAYER VOCABULARY

Player vocabulary refers to the knowledge and experience that the players gain during the process of playing the game. The purpose of player vocabulary is to enhance the skill of the players so that they are ready to take up more challenges. Another purpose is to define what the player can expect from the game. By setting up the expectation, you can avoid player frustration and disappointment and make the gaming sequence a memorable one. If the expectation is not properly set in the beginning, the excitement and surprises in the game will not give the players a good gaming experience.

Optimizing Game States

Optimizing the game states refers to the methods you employ to utilize every element of the game. When you utilize your objects and characters efficiently, you avoid overuse of game elements and duplication issues, ultimately reducing the memory requirement of the game. This ensures that your game runs smoothly across different platforms, ensuring player satisfaction.

CERTIFICATION READY
What is a game state?
3.2

A game state defines an object at a given point in time in the game. For example, in a car racing game, you might have a brand new car as well as a broken down car. You want to define the dissimilar properties for both the cars. Obviously, a player who gets the broken car needs to find the car malfunctioning. Therefore, you need to define the properties of this car to include flat tires, broken steering wheel, and so on. However, for a brand new car, the player must find the car functioning at its best and, therefore, you must define the properties accordingly. The game state allows you to make such definitions. You can create different game states and then define what each object in a game state does in that state.

You can use game states for almost everything in the game. You will need game states for the following:

- Creating menus
- Making multiple screens
- Using different levels in the game
- Killing a bad character
- Making a character walk, run, or perform other actions

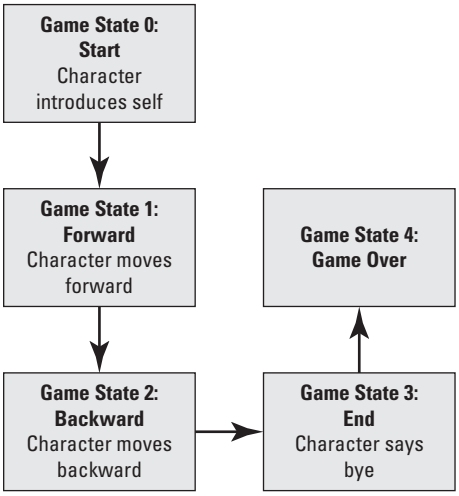
Let us learn about the concept of game states through a simple example. Imagine that your game goes through a set of actions:

1. The character on the screen introduces himself.
2. The character moves forward.
3. The character moves backward.
4. The character says “bye.”
5. The game ends.

Each of these actions provides the cue for the game to proceed further and, therefore, signals a game state. Figure 4-1 illustrates the game state sequence for the previous example.

Figure 4-1

The sequence of game states for a simple game



From the given example, you can infer that any game consists of a series of game states. The example assumes the value of the game state as numeric and that the value starts at zero. In each game state, a part of the game code is executed before the value of the game state variable is incremented to signal the commencement of the next game state.

TAKE NOTE*

In game programming, there are many ways to use a game state and even more ways to implement it in a game. You can manage game states using a simple Boolean value that is a binary variable with two possible values called "true" or "false"; or, you can even take advantage of complex polymorphism, in which you can use a class as more than one type through inheritance.

MANAGING GAMEFLOW

CERTIFICATION READY?
What is the role of game states in managing gameflow?
3.2

You can control the gameflow through game states. Game states are characterized by a combination of visual, audio, and animation cues that make the game proceed. To ensure that the game runs smoothly and contributes to a good gameflow, you need to optimize the game states.

Every element in a gameflow is defined for different game states. For your game to have a good gameflow, you need game states to capture the player's attention, sustain interest, and continue motivating the player at every point of time in the game. For each game state, you must provide something that is worthy of the player's attention. At the same time, you must ensure that you are not burdening the player with too many tasks in a given game state. In short, gameflow considerations can be implemented through game states. For example, suppose that in a gameflow, you want the game to become progressively challenging. In your game, the player character starts out having a range of weapons to choose from. For this to happen, in game stage 0, you define all the weapons to be enabled. However, to control the gameflow and to make it challenging, as you move to the next game state, you define only a few selected weapons to be enabled.

In this way, you create game states with customized definitions and operations for elements, such as challenges, scripted events, trial and error, player vocabulary, and other gameflow features. This allows you to control the gameflow to make the game challenging and interesting.

MANAGING PERFORMANCE

For your game to provide the best possible gaming experience, you need to make sure the game runs smoothly. But the heavy memory requirement of the game can pose challenges

to the game's smooth functioning. A game that uses many game elements or involves many actions happening at the same time makes the game memory-heavy because the game requires a lot of memory for processing these elements. As a result, performance issues might irritate the player and hamper the gameflow.

You need to utilize every element of the game consciously. This ensures that you avoid duplication, which in turn reduces the memory requirement. For example, consider that in a game state the player has to jump over a few stones. You need to create only as many stones as are required to pose an interesting challenge to the player. Too many stones not only irritate the player, but also make your game consume more memory. This consideration of how to best use each element in a game state is called *optimizing* the game state. Needless to say, each game state has to be optimized. Optimizing a game state requires you to ensure that every element in the game state does what it is supposed to do. This means that you need to specify the properties of each element in a manner that best utilizes that element. As a game developer, you must ensure that you optimize the game states to ensure that players get a chance to enjoy the game without any hindrances.

The most time-consuming task in a game is rendering. Rendering encapsulates the following concepts:

- Scene hierarchy
- Frame rate variations
- Graphics pipeline

To optimize the rendering process of your game, you must keep in mind the considerations regarding the previous concepts.

SCENE HIERARCHY

CERTIFICATION READY
How is scene hierarchy helpful in optimizing game states?
3.2

Scene hierarchy refers to the tree structure arrangement of logical and sometimes spatial representation of a graphical scene. An operation performed on a group of nodes in this structure affects all its members. For example, if the light source for a scene changes from daylight to sunset ambience, then all the objects in the game world that are illuminated by daylight will change to reflect the effect of the revised light.

Scene hierarchy is useful for optimizing the rendering and for collision detection, which in turn optimizes the frame rate. The higher the frame rate, the smoother the gameplay—that is, the visual, audio, and animation cues are also smooth. Therefore, scene hierarchy is an important consideration in optimizing game states. It helps you to manage the memory allocation and, therefore, helps you design a game that requires less memory to run smoothly.

Scene hierarchy uses instancing to reduce memory costs and increase speed. This is possible because of the parent-child tree structure of a scene hierarchy. The nodes in a scene hierarchy stand for objects in a scene.

For example, a character riding a horse can be defined in the game to have a logical relationship with the horse (see Figure 4-2). In such an instance, the scene hierarchy has two nodes, one each for the character and the horse. It also specifies the movement of the character as the horse moves in 3D space.

You must use scene hierarchy to optimize the game state when:

- You need to frequently create and destroy objects
- You have similar objects in a scene
- Each object has a resource that is slow to acquire and can be reused

Figure 4-2

A game character riding a horse

©Microsoft Corporation

FRAME RATE VARIATIONS

CERTIFICATION READY?
How do frame rate variations affect the player experience in a game?
3.2

You already know that frame rate is the rate at which an image is refreshed and this rate is expressed in ***frames per second (fps)***, which is the number of frames displayed per second by a display device. Usually, other processes run behind the scene when an image is refreshed. These include collision detection and network processing.

Like scene hierarchy, fps also affects player experience. It affects game performance in two ways:

- Low fps makes motion appear with considerable delay, which affects the player's capacity to interact with the game.
- An fps that varies a lot between seconds due to the load of heavy computation makes the motion seem choppy or jagged.

For games that require players to track animations and react quickly, fps-induced delays or hindrances to members disrupt the gameflow. Therefore, you must consider fps while creating an optimized game state. Although more and more game developers are adopting a higher value of fps, it might not be a practical choice in all instances. This is because though a higher fps might guarantee a faster motion, it cannot remove an input lag. Input lag is the delay between player command and computer feedback. A higher fps does not also guarantee fluid movements.

GRAPHICS PIPELINE

CERTIFICATION READY?
What is a graphics pipeline?
3.2

A ***graphics pipeline*** refers to the current method of drawing images on a screen through a conversion of vector graphics format or geometry into raster or pixel images. The 3D geometry is given as input at one end of the pipeline; the 3D engine processes the geometry and gives the output as a 2D image that is presented on the screen. Using the 3D pipeline does not optimize gameplay. You will need to optimize each stage of the pipeline.

The stages in a graphics pipeline and the questions answered at each stage are:

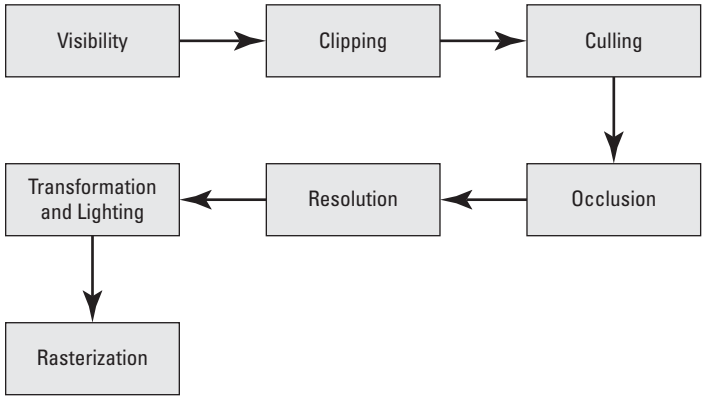
- **Visibility:** What should display on the screen?
- **Clipping:** What is in the viewing area of the camera?

- **Culling:** Is the image to be placed facing the camera?
- **Occlusion:** Is the object hidden from the camera's point of view?
- **Resolution:** What is the resolution of the display?
- **Transformation and lighting:** How will the objects and characters in the game be moving, scaled, lighted, and so on?
- **Rasterization:** How will the gameplay elements be drawn on the screen using pixels?

Figure 4-3 displays the graphics pipeline in a process flowchart.

There is a direct relationship between the details in the 3D geometry and the amount of memory required to perform the operations on the 3D geometry. Similar to scene hierarchy, the graphics pipeline also needs to be considered if you want to reduce the memory required for the smooth running of the game.

Figure 4-3
The graphics pipeline flowchart



Defining the Game Loops

To ensure that you create a game that runs smoothly, you must define the main loop, or a part of code, that gets repeated throughout the duration of the game.

CERTIFICATION READY?
What are the different methods available in the XNA Framework for a game's main loop?
3.2

At the heart of a game's code is its main loop (or ***game loop***), which is responsible for coordinating the actions of the game's program. The main loop is similar in its functioning to that of the nervous system of our body. Both relay signals and messages to and from different parts of the system they are part of.

In its basic form, the main loop performs two functions:

- **Executes the actual gameplay:** This function involves moving objects, changing the game's status and levels, and all the other actions that constitute the actual game. This is the core function of the game, and it executes in a loop.
- **Draws the game components:** This function involves drawing game objects like static models, animated models or characters, and sprites on the screen. This helps the player to view all the potential game components on the screen. This function executes in a loop.

Because a game loop takes in player input, game loops influence the game states. A game state can change depending on the user input. The game loop, therefore, plays an important part in capturing the user input and updating the game state.

It is advantageous to use a game loop. Computer systems run on the principle that unless the user provides an input, the software does not need to do anything. However, games must run continuously whether or not the player makes a move. You can use game loops to continue the game. A simple pseudo code for a main loop can be:

```
While (player does not exit)
{
        Update player position according to input from keyboard
or mouse or joystick.

        Update AI according to player position.
        Update Physics and Sound Engine.
        Calculate culling and clipping.
        Set proper shader to objects and set Render states.
        Draw game objects.
}
```

The disadvantage of using a basic main loop is that it does not handle time, and a game requires time-based procedures for movement and animation. If we do not handle time, the game will run slower or faster depending on the game platform. Loops for console games can utilize the processing resources easily. However, loops for games on Windows-based computers and consoles need to operate within the limits set by the process scheduler. Generally, all games are made to run at a fixed fps—usually at 60 fps.

By default, the XNA Framework's `Game` class manages a main loop. The XNA Framework provides the following methods for the main loop to interact with the game platform:

- `Update`
- `Draw`

Let us look at each of these methods in detail.

UPDATE METHOD

The `Update` method is called whenever the game logic needs to be processed. This method takes an input parameter of `GameTime` type. This parameter gives you the time that has passed since the last call to the `Update` method.

You require the code to call the `Update` method for:

- Managing the game state
- Processing player input
- Updating simulation data

You need to override this method for including the game logic that is specific to your game. The following is the skeletal structure of the `Game.Update` method.

```
protected override void Update (GameTime gameTime)
{
        /* enter your game logic here */
        base.Update(gameTime);
}
```

DRAW METHOD

The `Draw` method is called whenever it is time to draw a frame. You need to override this method with game-specific code for drawing frames. This method takes an input parameter of `GameTime` type. This parameter gives you the time passed since the last call to the `Draw` method.

The following is the skeletal structure of the Game.Draw method:

```
protected override void Draw (GameTime gameTime)
{
/* perform game-specific rendering here */
base.Draw(gameTime);
}
```

TYPES OF GAME LOOPS

How often the Update method is called in a game is defined by the type of game loop used. There are two types of main loops:

- Fixed step
- Variable step

In the ***fixed step game loop***, the game code calls the Update method at fixed intervals. You can specify this interval in the Game.TargetElaspedTime parameter declaration. In the XNA Framework, the default for Game.TargetElapsedTime is 1/60th of a second. The Update method will be called once the Game.TargetElapsedTime value is met. After executing the Update method, if it is not time to call Update again, the code will move on to the Draw method. If Update takes longer and it is time to call the Update method again, the code will not call the Draw method. This means that the game can slow down and might irritate the player. For example, imagine a situation in which, based on the player's input, the player character must show some movement on the screen. However, when the Game.TargetElaspedTime is met even before the player input can be captured, the Draw method is not called and, therefore, although the player is entering the key to make the character move, he or she cannot see the character move on the screen.

In the ***variable step game loop***, the game code calls the Update and Draw methods continuously regardless of TargetElapsedTime. This means that the player can see the character move on the screen regardless of whether the TargetElapsedTime has been met or not. Animations and movement operations are based on precise timing; therefore, the variable step game loop is suitable for games with animations and movement operations. A variable step game loop takes the GameTime.ElapsedGameTime parameter on which the game logic and animation code is based. The GameTime.ElapsedGameTime is the amount of time elapsed since the last update.

In XNA, you can make your game use a variable time step by setting the IsFixedTimeStep property of the Game class to false. The following code sets the Game.IsFixedTimeStep property to false in the Game.Initialize method.

```
protected override void Initialize()
{
this.IsFixedTimeStep = false;
base.Initialize ();
}
```

To make a sprite move consistently when you use a variable step game loop, you need to specify the distance the sprite moves in terms of game units per unit time. The amount a sprite moves in an update will then be automatically calculated as the rate of sprite movement multiplied by the elapsed time.

Table 4-1 presents the difference between the two game loops at a glance.

Table 4-1

Difference between Fixed Step and Variable Step Game Loops

Type of Game Loop	Features	Pros	Cons
Fixed Step	• Assumes each update is for a fixed time • Position of a sprite = position of the sprite + distance moved per update	• Easier to calculate the physics associated with movement • Easier to record actions per time step	• Not suitable for time-based animations • Minor physics errors are introduced that can multiply into big coding flaws
Variable Step	• Calls `Update` and `Draw` continuously • Position of a sprite = position of the sprite + distance moved per update* timeElapsed	• Easier to code • Smooth running of game	• Hard to replay or record actions as time steps vary • Minor physics errors are introduced that can multiply into big coding flaws

■ Designing Objects and Characters

THE BOTTOM LINE

Well-designed characters and objects make the game world realistic for players. Most action, adventure, and role-playing games depend extensively on the popularity of well-known characters, not only for entertainment and story-building, but also for ensuring the success of the game.

Most gaming projects have their own visual design team. The main responsibility of this team is to use its artistic skills to ensure that the design of characters or objects fits well into the visual design of the game.

Once the objects and characters for a game are designed and illustrated, the visual design team needs to draw them using the appropriate 2D and 3D software, such as Autodesk Maya, Autodesk Max, Autodesk XSI, Modo, and so on. Once objects are created, you should export them to XNA Framework 4.0 using the specific file formats such ".x" or ".fbx".

The objects and characters are then further enhanced by applying textures or adding a light source. Lighting is a great medium for further developing the look and feel of the environment and the characters in the game. It makes the objects and characters come alive during the gameplay.

Whatever you see on the screen is always in 2D because the computer screen does not have three dimensions. You will use the 2D images or 3D geometry information of the visual images for procedures such as:

- Transforming objects
- Animating the characters
- Creating the feel of the character

Transforming Objects

Transformation means converting the geometrical vertices of objects and characters to proper positions on the screen. The transformation process involves a set of activities that are performed on these vertices in relation to three spatial references: world, view, and projection.

When objects are created on any 3D software, they are created in object space. In the object space, you initially create your object by setting (0, 0, 0) as the center of the object and defining the vertex positions relative to the center. You need to place these objects within the game world at some distance or at some angle with respect to each other, which is called transforming objects from object space to world space. You need a matrix to transform objects in the XNA Framework. A matrix is a two-dimensional grid of numbers that helps to transform coordinates of a model in the object space into coordinates in the game world. The following shows a 4×4 matrix.

$$\begin{bmatrix} M11 & M12 & M12 & M14 \\ M21 & M22 & M23 & M24 \\ M31 & M32 & M33 & M34 \\ M41 & M42 & M43 & M44 \end{bmatrix}$$

TAKE NOTE*

In geometry, a coordinate system is a system that uses coordinates that uniquely determine the position of a point or other geometric object. In XNA Framework, the objects are represented using a right-handed Cartesian coordinate system. That means the positive Z-axis points toward the observer when the positive X-axis and the positive Y-axis point right and up, respectively.

MATRICES IN XNA

CERTIFICATION READY
What are the different types of matrices that the XNA Framework 4.0 uses for transformation?
4.1

The XNA Framework requires three different matrices for transformation before the three-dimensional model is rendered on the 2D screen. You must first understand the basic transformation that happens when an object or a model is rendered on the screen. The following are the sequence of actions of the geometry of a model before the coordinates of the model are transformed into 2D pixels on the screen.

1. First, you create the 3D model in an editor such as Maya and define the model by keeping (0, 0, 0) as its center. The coordinates of the vertices are with respect to the center of the model. This is called the object space.
2. To transform the coordinates from the object space to the game world, known as the ***world space***, you need to provide a ***world matrix***. The XNA uses this world matrix to convert the vertices in the object space to the world space. The coordinates of the world space are with respect to the whole game world. The world matrix tells the position of the model in the game environment. You need to calculate the world matrix before you render any object in the game world.
3. The vertices in the world space need to be transformed so that they lie in the camera space known as the ***view space***, which is the line-of-sight of the player or the player character. To achieve this, you need to provide a matrix, known as ***view matrix***, that

helps the XNA convert the vertices in the world space to the view space. The view matrix tells the position of the player with respect to the game world. You need to calculate the view matrix whenever the camera changes its view or position.

4. Finally, the XNA needs to project the 3D coordinates in the view space onto the flat 2D screen, which is called the ***projection space***. For this, you need to provide a matrix called the ***projection matrix*** to the XNA. This matrix helps the XNA convert the 3D coordinates into screen coordinates. You normally set the projection matrix only once at the time of initialization. This transformation tells the position on the screen where the vertices of the model should appear.

TAKE NOTE* To animate an object, you transform the coordinates of the object for every frame. This transformation needs to be in relation to three spatial references, namely, the world space, the view space, and the projection space.

The transformation of the coordinates usually happens in the graphics pipeline. However, you need to create game-specific code for the desired animation.

The XNA Framework 4.0 provides the matrix structure to help you to create the world matrix, view matrix, and the projection matrix. The XNA matrix is a 4 × 4 matrix. Table 4-2 lists some of the methods available in the matrix structure.

Table 4-2
Public Methods of the Matrix Structure

Method	Description
`CreateLookAt`	This method creates a view matrix. You need to set the view matrix for an object whenever there is a change in the camera position.
`CreateWorld`	This method creates a world matrix.
`CreatePerspectiveFieldofView`	This method creates a perspective projection.
`CreateTranslation`	This method creates a translation matrix, which when applied to the geometry of the model, transforms it into the world space.

VECTORS IN XNA

You use vectors to represent a direction and the scale of geometry. The XNA Framework 4.0 provides classes, namely `Vector2`, `Vector3`, and `Vector4`, to represent a vector or the coordinates of a vertex. The `Vector2` class represents a vector with two components (X and Y). The `Vector3` class represents a vector with three components (X, Y, and Z). The `Vector4` class represents a vector with four components (W, X, Y, and Z). Tables 4-3, 4-4, 4-5, and 4-6 list some of the properties and methods of these vector classes.

Table 4-3
Some of the Public Properties of the `Vector2` Structure

Field	Description
`One`	This property returns `Vector2` with one in all of its components.
`UnitX`	This property returns X-unit `Vector2` (1, 0).
`UnitY`	This property returns Y-unit `Vector2` (0, 1).
`Zero`	This property returns `Vector2` with zero in all of its components.

Table 4-4

Some of the Public Properties of the `Vector3` Structure

Field	Description
`One`	This property returns `Vector3` with one in all of its components.
`UnitX`	This property returns X-unit `Vector3` (1, 0, 0).
`UnitY`	This property returns Y-unit `Vector3` (0, 1, 0).
`UnitZ`	This property returns Z-unit `Vector3` (0, 0, 1).
`Zero`	This property returns `Vector3` with zero in all of its components.

Table 4-5

Some of the Public Properties of the `Vector4` Structure

Field	Description
`One`	This property returns `Vector4` with one in all of its components.
`UnitX`	This property returns X-unit `Vector4` (1, 0, 0, 0).
`UnitY`	This property returns Y-unit `Vector4` (0, 1, 0, 0).
`UnitZ`	This property returns Z-unit `Vector4` (0, 0, 1, 0).
`UnitW`	This property returns W-unit `Vector4` (0, 0, 0, 1).
`Zero`	This property returns `Vector4` with zero in all of its components.

Table 4-6

Some of the Methods of the Vector Structures

Method	Application
`Dot`	This method calculates the dot product two vectors.
`Normalize`	This method creates a unit vector from the specified vector.
`Transform`	This method transforms the given vector class or an array of vectors by a specified matrix.

Transforming objects is the initial step for various operations, such as:

- Forming objects
- Deforming objects
- Moving objects
- Inserting point distance between objects
- Creating planes
- Modifying keyframe interpolation

FORM OBJECTS

CERTIFICATION READY
How can you form your game objects in XNA Framework?
4.2

Every object in computer graphics can be split into triangles. A triangle is the primitive part of any shape. For example, a box or even a circle can be formed using triangles. To form 2D images of the 3D objects you create for the game, you need to break the object into triangles. You then input the 3D triangle information in the graphics pipeline and then the XNA Framework converts the information to draw the 2D image.

The XNA Framework 4.0 provides primitive type enumeration, which you can use to draw primitive shapes such as triangles. The Framework also provides methods such as `DrawUserPrimitives`, `DrawUserIndexedPrimitives`, and `DrawPrimitives` in the `GraphicsDevice` class to render the primitive geometry. These methods use the primitive

TAKE NOTE*

Primitive shapes other than triangles are lines, points, and polygons.

type enumerations and tell the graphics device how to interpret the vertices stored in a vertex array. The enumerations include:

- `PrimitiveType.LineList`
- `PrimitiveType.TriangleList`
- `PrimitiveType.LineStrip`
- `PrimitiveType.TriangleStrip`

Now let us see how to draw a triangle using the `PrimitiveType.TriangleList` enumeration in the following steps. The code in the step uses the `GraphicsDevice.DrawUserPrimitives` method to render the triangle.

DRAW A TRIANGLE

GET READY. To draw a triangle, perform the following steps:

1. Declare the following variable in your Game class. The `VertexPositionColor` used in the code is a structure to hold position and color information of a vertex. The code declares an array of the `VertexPositionColor` structure as you need three vertices to draw a triangle.

```
        /*Declare vertex properties */
    VertexPositionColor[] vertices;
    /* The BasicEffect is an built-in class in XNA which is
 used to contain basic rendering effect.*/
    BasicEffect basicEffect ;
```

2. Create a method called `InitializeVertices` in the Game class to define the properties of the vertices.

```
                public  void Initializevertices ()
    {
/*Defines the vertex properties such as the position and color
   information associated with each Vertices. We need this
   information to draw and update the triangle */
vertices = new VertexPositionColor[3];
/* Position and color info for 1st vertex .*/
   vertices[0].Position = new Vector3(-0.5f, -0.5f, 0f);
   vertices[0].Color = Color.Red;
/* Position and color info for 2nd vertex .*/
   vertices[1].Position = new Vector3(0, 0.5f, 0f);
   vertices[1].Color = Color.Green;
/* Position and color info for 3rd vertex .*/
   vertices[2].Position = new Vector3(0.5f, -0.5f, 0f);
   vertices[2].Color = Color.Yellow;
/*Now define your BasicEffect object as its going to be used
   while drawing the triangle.*/
basicEffect = new BasicEffect(GraphicsDevice);
}
```

3. Override the `LoadContent` method by calling the `InitializeVertices` method.

```
protected override void LoadContent()
{
    Initializevertices();
}
```

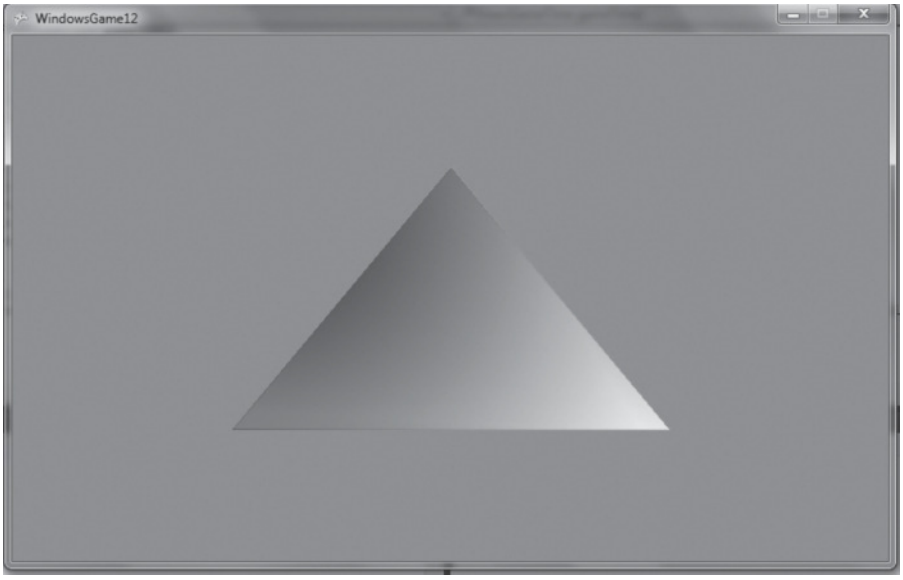
4. Override the Game.Draw method to draw the triangle as in the following.

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

/*Set the culling state for Device to CullMode.None so that
   triagle will always be visible*/
RasterizerState state = new RasterizerState();
             state.CullMode = CullMode.None;
             GraphicsDevice.RasterizerState = state;

//Set the basic effect parameter to show the color of each
   vertex.
             basicEffect.VertexColorEnabled = true;

/*Now iterate through all passes of currentTechnique of basic
   effect and draw the triangle */
foreach (EffectPass pass in basicEffect.CurrentTechnique.Passes)
{
                pass.Apply();
                /* Draw the triangle using DrawUserPrimitives
  function of the GraphicsDevice*/

  GraphicsDevice.DrawUserPrimitives<VertexPositionColor>
  (PrimitiveType.TriangleList, vertices, 0, 1,
  VertexPositionColor.VertexDeclaration);

  }
}
```

Figure 4-4 shows the game screen displaying the triangle, which is the output of the previous code.

Figure 4-4

A game screen displaying the triangle



You can create a human character using the methods listed, but it takes a lot of time. `TriangleList` and `LineList` are not practical methods for creating characters, so they are not used for this. As already discussed, the visual design team creates a 3D model of a game character using AutoDesk Maya or other similar modeling software. You need to export the model using a file format such as .x or .fbx to use them in the XNA environment. You can then use the `Model` class available in the XNA Framework to load the human character directly from this 3D modeling software. The following code assumes that the required game model, a car in this case, is exported into the specified project from the appropriate 3D environment in the .x format. To export a model into the XNA environment, perform the following steps.

1. Right click the Content node in the Solution Explorer. Then click Add > Existing Item as shown in Figure 4-5.
2. In the Add Existing Item dialog box that is displayed, select the required model in the specified file format as shown in Figure 4-6.

Figure 4-7 shows the car model used in this example. The code loads the 3D car model into the game world at run time. You can include the given code in the `Game.LoadContent` method:

```
Model currentModel = Content.Load<Model>("Car.fbx");
```

In the `Game.Draw` method, render the car model using the following code:

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

        //Initialize View matrix for Drawing the object

        Matrix viewMatrix = Matrix.CreateLookAt(new
Vector3(0.0f, 20.0f, 20.0f),Vector3.Zero, Vector3.Up);

        /*Initialize projection matrix for Drawing Object*/
        Matrix projectionMatrix = Matrix.CreatePerspectiveFieldOf
View((float)Math.PI / 4.0f, 1.33f, 0.01f, 1000.0f);
```

Figure 4-5

The content node displaying the Add menu

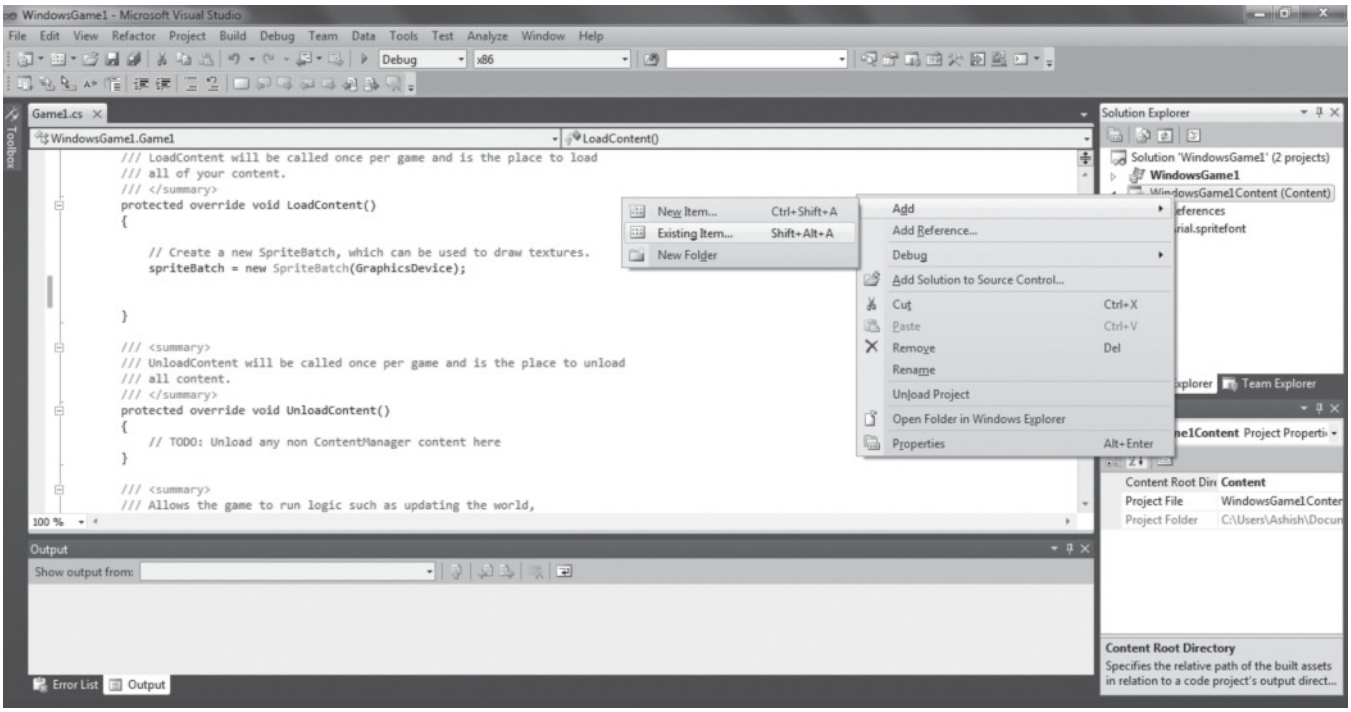


Figure 4-6

The Add Existing Item dialog box

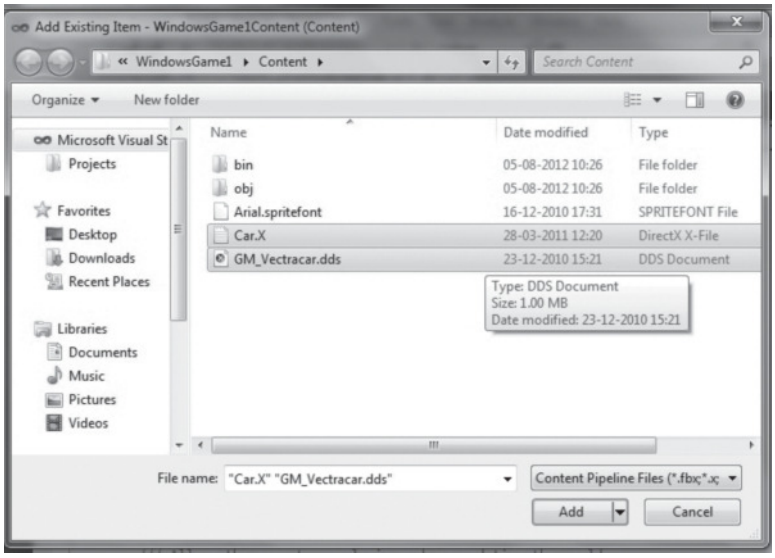


Figure 4-7

The car model


```
            /*Set the Culling mode so that unnecessary triangles will
        not be processed . CullMode is defined as a member of RasterizeState
        in XNA which defines the current render state of Device*/
            RasterizerState state = new RasterizerState();
            state.CullMode = CullMode.CullCounterClockwiseFace;
            graphics.GraphicsDevice.RasterizerState = state;
            currentModel.Draw(Matrix.Identity, viewMatrix,
projectionMatrix);
    }
```

CERTIFICATION READY
What is deformation of a game object?
4.2

DEFORMING OBJECTS

Sometimes, you will need to change the shape of a game model from its original shape as a result of an event. For example, in a car racing game, assume that a player character's car has crashed. In this case, you need to show the impact of the crash by deforming the area of the car affected by the crash. For instance, you can show a dent in the front bumper. In this way, you can provide a convincing look of damage.

Deformation generally finds uses in games. You can use deformation, for example, to make a character fat or thin, to create ocean waves, or simply to change the appearance of an object. Whatever the end result, to deform the object, you have to manipulate the geometry and the texture information. To do this, you change the 3D geometry information of the object and then send the revised information for transformation into 2D pixels.

Deforming objects is made easy by the use of shaders.

XREF
For more information about shaders, refer to the section, "Creating the Feel of the Character," in this lesson.

CERTIFICATION READY
How will you move your game objects?
4.2

MOVING OBJECTS

All games contain both static objects and moveable objects. Static objects in your game can be trees, stones, walls, and so on. However, movement of static objects can also change in accordance with the game-specific requirements. For example, in a maze game, the walls can move, or in a car-racing game, the stones can move as a car races by.

Moveable objects in a game can be human characters, animals, or vehicles. The moveable objects, as the name suggests, need to be in motion depending on the game storyline.

When you move objects, you need to essentially provide the XNA Framework with the changed position vertices of the object, and the graphics pipeline will then transform the motion on the screen. However, you should consider other elements when you move the object. For example, you might need to consider whether the object will collide with another object on the screen, or whether the light effect on the object needs to be revised.

The following code sample moves an object:

```
protected override void Update(GameTime gameTime)
{
                //Update  position in every frame
        Vector3 position = modelInitialPosition + 3.0f
*gameTime*anyDirection ;
                //Create translation matrix from that position
        worldmatrix = Matrix.createTranslation(position);
}

protected override void Draw(GameTime gameTime)
{
        //Initialize effect and draw model
        BasicEffect effect = new BasicEffect();

        effect.World = worldMatrix ;

        model.Draw();

}
```

CERTIFICATION READY
Why do you need to calculate the point distance between objects?
4.2

INSERTING POINT DISTANCE BETWEEN OBJECTS

When designing the objects in a game, it is essential to calculate the constant movement transitions of the included objects. An object takes some time to move from one point to another; a movement necessitates that you must account for the distance covered.

Keeping track of the distance covered allows you to properly position the object on the screen, avoiding collisions with other objects. It also allows you to maintain a constant speed of movement.

For example, imagine a game scenario in which a player is playing against a robot. If the player fires a bullet on sighting the robot, the animation of the bullet on the screen must imitate real life. As such, it needs to travel the distance and path as per ballistic physics. The calculation depends on the kind of weapon used by the player. Depending on the calculation, an associated kill range for the bullet can be determined. If the robot falls in the kill range, the player will be successful in killing the opponent.

You need to calculate distances for performing operations, such as moving, rotating, and scaling your objects within a plane. In the XNA Framework, you can use the built-in `Vector2.Distance` function to calculate the distance between two objects. You can also use the `Distance` method of the `MathHelper` class to calculate the distance and analyze it against the time taken to ensure constant speed of object movement.

CREATING PLANES

CERTIFICATION READY
What is the importance of planes in transforming objects?
4.2

A discussion on transformation of objects is not complete without mentioning planes and associated transformations. In the XNA Framework, you can perform operations such as rotation, translation, and scaling only when the object is defined within a plane.

When defining a plane, you need to specify the normal vector, which is the perpendicular vector to the plane, and its minimum distance to the coordinates of the origin of the plane. In 2D and 3D structures, the coordinates are represented as (0, 0) and (0, 0, 0), respectively. So, to create a plane, you require its angle of rotation (normal direction) and height (distance to the points on the coordinate axes). Every plane has a unique combination of normal and distance values. Figure 4-8 illustrates the plane as represented by the XNA Framework.

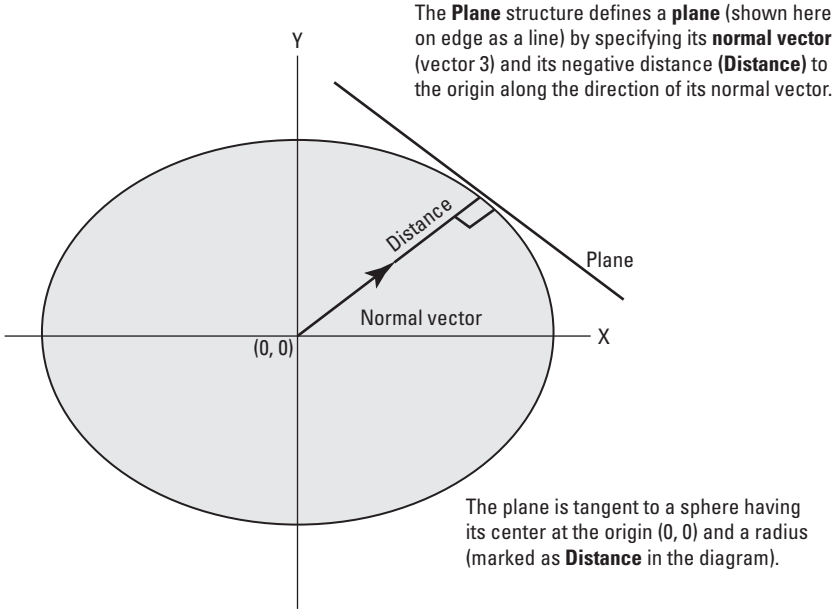


Figure 4-8
The plane structure in the XNA Framework

In the XNA Framework, we use the `Plane` structure to define a plane. The `Plane` structure provides fields and members. Table 4-7 and Table 4-8 list the fields and methods of the `Plane` structure, respectively.

Table 4-7

Public Fields of the Plane Structure

Field	Description
D	This field provides the distance of the Plane along its normal vector from the origin.
Normal	This field provides the normal vector of the Plane. The normal vector is a Vector3 type.

Table 4-8

Methods of the Plane Structure

Method	Application
Dot	Calculates the dot product of a specified Vector4 and the current plane.
DotCoordinate	Returns the dot product of a specified Vector3 and the Normal vector of the current plane along with the D constant value of the plane.
DotNormal	Returns the dot product of a specified Vector3 and the Normal vector of the current plane.
Equals	Determines whether two instances of a plane are equal.
Intersects	Checks whether a plane intersects a bounding volume.
Normalize	Changes the coefficients of the Normal vector of a plane to make it of unit length.
Transform	Transforms a normalized plane by a matrix or quaternion.

XREF

For more information on rotating an object in a plane, see the section, "Scaling and Rotating Matrices," later in this chapter.

TAKE NOTE*

Three concepts—yaw, pitch, and roll—are the angles to measure the movement of an object that helps to understand the rotation of an object in a plane. Yaw means rotating at Y-axis that points up, pitch means rotating at X-axis that points right, and roll means rotating at Z-axis that points towards the observer. The XYZ are orthogonal to each other.

You can use matrices to perform any of these rotations. Matrices are also used for scaling and translation operations on an object in a plane.

CERTIFICATION READY
What is keyframe animation?
4.1

MODIFYING KEYFRAME INTERPOLATION

TAKE NOTE*

You can create key-frame animations using Silverlight as an XAML file.

You generally present animation as a series of images. However, if there is some processing delay, your game could end up showing jagged animations. To prevent this, you must use the interpolation concept and provide in-betweens, or frames that lie between a start frame and an end frame. ***Interpolation*** refers to the method of constructing new data points in between the two separate sets of defined data points. In-betweens are frames constructed between two keyframes.

Keyframe animation focuses on the start and endpoint of an animation and consists of many frames between these two points. You create the keyframes, and the XNA Framework enhances the display with the in-betweens.

MORE INFORMATION

For more information about creating keyframe animations, refer to the "Keyframe Animations" section in the MSDN Library.

Each keyframe includes parameters such as position, rotation, and scales of an object. To create in-betweens, the XNA Framework takes the parameters from the two keyframes and then predicts the positions that lie within the probable path of animation.

For example, imagine animating a spaceship moving from the upper-left corner of the screen to the lower-right corner of the screen. The distance is known and can be expressed by changing the matrix applied when drawing the spaceship. If the matrix is changed in a single update operation, though, the spaceship jumps from one corner to the other. A smooth animation requires movement across multiple frames, each individual movement smaller than the whole corner-to-corner move. Rather than specifying each intermediate position, the intermediate locations are interpolated by the system for each frame.

When transforming objects, you might want to display the transformation on the screen through animation. For example, suppose a car crashes and you want to show the dents on the car being formed in an animation. To bring realism to the animation, you must modify the keyframe interpolation of the images.

You can modify the keyframe interpolation through specific methods in the XAML file. A keyframe interpolation method of an animation describes the transition of the animation between the two keyframes during a period of time. You can define the keyframe interpolation method with reference to the animation. So, the type of interframe depends on the interpolation method that it uses. Table 4-9 lists the three interpolation methods and their features.

Table 4-9

Keyframe Interpolation Methods

Interpolation Method	Feature	Example of Spaceship Animation
Linear	The individual interframes move with constant speed.	The spaceship traverses diagonally at a constant speed.
Discrete	The animation moves from one keyframe to the next without any interpolation. In such cases, no interframe is created and the animation consists of showing only the two keyframes on the screen.	The spaceship disappears from the first corner and appears at the second corner.
Splined	The animation displays realistic movement of objects within the specified duration. The KeySpline property makes its function unique when compared with the other keyframes. The animation with the splined keyframe proceeds quickly at the start, then slows down in between, and then again speeds up towards the completion.	The spaceship moves quickly at the start, slows towards the center, and then speeds up towards the second corner, giving a realistic feel of an actual spaceship in movement, slowing occasionally to avoid accidents on the route.

You can also create keyframe animations using AutoDesk Maya or AutoDesk Max or any 3D Software. Once you create keyframe animations, you need to export them to .fbx format. You can then load the specified model in .fbx format into the XNA Framework using the following code. You need to provide the following code in the `Game.LoadContent` method:

```
Model mesh = Content.Load<Model>(<Filepath>);
```

Animating the Basic Character

When designing a character for a game, it is important to make the character lively under the given set of conditions.

After modeling a character, you need to animate it. Animation is essentially a series of frames or images. To animate your character in XNA, you need to give your character movement, set up the rate at which frames of images will be displayed on the screen, which will in turn control the speed of animation, and provide directions for animation effects. You use the techniques described for transforming, moving, deforming, and other such operations in the previous section to animate the character.

Animating the basic character involves:

- Movement
- Frames per second (fps)
- Sprite animation
- Scaling and rotating matrices

Let us see each of them in detail.

MOVEMENT

When animating a character, you must carefully design every action or movement. Movement of a character, especially of a living being, differs from the movement of other objects in a game.

In general, animating a character involves rotation and movement or translation of the character. Now, let us see how to animate a game model using code samples.

The following code sample declares an instance of the `Model` class and loads the game model into the `Model` object.

```
      /* Declare the 3D model and animation player to play animation
in the Game class. AnimationPlayer is an built-in class in XNA which
is used to play animation.*/
            Model currentModel;
            AnimationPlayer animationPlayer;
            Matrix cameraViewMatrx;
            Matrix projectionMatrix;
            Vector3 Position;

      /* Now load the model and get the required information to play
an animation in the LoadContent method of the Game class */

     protected override void LoadContent()

     {

     //Load model

     currentModel = Content.Load<Model>(your model imported from
     Maya in fbx format);

     /*Now retrieve skinning data . We need to get the skinning
     information as it is required to initialize the AnimationPlayer
     object which eventually will play the animation .*/
     SkinningData skinningData = currentModel.Tag as SkinningData;
```

```
//initialize animation player which will be used to play animation

animationPlayer = new AnimationPlayer(skinningData);

/*get desired animation information using the AnimationClip
object */
              AnimationClip clip = skinningData.
AnimationClips["idle"];
//play animation
              animationPlayer.StartClip(clip);
/*initialize view and projection matrix for drawing the model*/
cameraViewMatrx = Matrix.CreateLookAt(new Vector3(0.0f, 20.0f,
20.0f), Vector3.Zero, Vector3.Up);

/ * Initialize projection matrix for Drawing Object */
projectionMatrix 5
Matrix.CreatePerspectiveFieldOfView((float)Math.PI / 4.0f, 1.33f,
0.01f, 1000.0f);

}
```

Once you play the animation, you need to update it every frame. You can do it by calling the `Update` method of the `AnimationPlayer` object in the `Game.Update` method. The following code runs the animation:

```
protected override void Update(GameTime gameTime)
{
//Update animation with time
animationPlayer.Update(gameTime.ElapsedGameTime, true, Matrix.
      Identity);
}
```

Now, you need to draw the model to check whether animation is playing or not. This part is bit tricky. To understand this, you need to know some basics of animation. In real life, as bones in the body move, muscle and skin move along with it. Similarly in the 3D environment, playing the animation changes the bone of the character. In the `Game.Draw` method, you retrieve this bone information and move the parts of the character using the transformation matrix of the associated bone. The following code draws the model:

```
Protected override void Draw (GameTime gametime)

{
        //Set bone matrix

        Matrix[] bones =
   animationPlayer.GetSkinTransforms();

        //Iterate each mesh part in the model

              foreach (ModelMesh mesh in currentModel.Meshes)

              {

                    /*iterate through each effect*/

                    foreach (SkinnedEffect effect in mesh.
                    Effects)
```

```
                                {

                                    //Set bone information

                                    effect.SetBoneTransform
                                    s(bones);
                                    /*Set other tranformation matrices */
                                    effect.world = Matrix.Identity;

                                    effect.View = cameraViewMatrx;
                                    effect.Projection =
        projectionMatrix;

                                           effect.EnableDefault
                                    Lighting();

                                }

                                mesh.Draw();
                                }
        }
```

To rotate the character simultaneously, declare the following variable in your `Game` class:

```
Matrix rotation ;
```

Now initialize the matrix in the `Game.Initialize` method:

```
/* Declare a matrix object in Game.Initialize Method */
rotation =Matrix.Identity;
Position =Vector3.Zero;
```

Now in the `Game.Update` method, create a rotation matrix on the key press of "D":

```
if (keyState.IsKeyDown(Keys.D))
{
      rotation *=
Matrix.CreateFromAxisAngle(Vector3.Up, -MathHelper.Pi / 25.0f);
}
```

In the `Game.Draw` method, set the world matrix to rotation matrix:

```
/* In the Game.Draw() method set worldmatrix to
rotation before rendering */
      effect.World = rotation ;
```

You can also translate the character while walking. First, you play "walk" animation of the character, and then modify the `AnimationClip` object as shown below.

```
AnimationClip clip =
skinningData.AnimationClips["walk"];
```

In the `Update` loop, on any key press, provide the following code:

```
Position += Vector3 (0, 0, 2.0f * gameTime);
```

In the `Draw` method, provide the following code:

```
effect.World = Matrix.CreateTranslation(Position);
```

An artist provides the animation names such as "idle" and "walk" that are used in the code. An animator can set the name of animation, while animating the character in Max, Maya, or any 3D software. He can also set the animation name, while exporting the animation to any XNA specific format. To know the animations available in any exported object, you can open the .x or .fbx file in a Notepad application and check for animation names.

FRAME RATE

CERTIFICATION READY
How does fps affect animation?
4.1

As already discussed, frame rate is the speed at which the image is refreshed. A frame refers to a single image produced by the imaging or a display device. *Frames* refers to unique consecutive images and *second* just refers to the unit of time in which they are displayed. For example, an animation that is locked at 12 fps takes one second to view every 12 consecutive unique images.

A frame rate affects the animation in video games in two ways. First, when fps is set low, it does not effectively produce the illusion of movement. Second, when fps is not consistent due to increased time in the computation of a frame and differs largely from one second to the next, it produces a jerky motion or animation.

The frame rate can vary from game to game. It depends on the style of the game. An action game might require frame rates locked between 30 fps and 60 fps, because these games involve the players to track the animated characters or objects quickly and react faster. This also helps changeover the frames so quickly that it will not be easily visible and will appear as if the character performs realistically.

SPRITE ANIMATION

CERTIFICATION READY
How do you animate sprites?
4.1

Generally, when you animate an object, you make the required changes to the position, appearance, and other modifications manually to each image. But consider a situation in which you need to make a similar change on a number of copies of the same object. Just thinking about it makes you feel exhausted.

Essentially, ***sprite animation*** works on the principle that animation in 2D games can be set by creating a group of images that vary only slightly from each other. The differences in the images should be such that the human eye cannot distinguish these dissimilarities.

To create the sprite animation of a spaceship, for example, you first prepare the bare minimum number of frames required for animation. You then make these frames move around one after another in a continuous motion. You do not need to manually redraw the frames for each loop.

The following is the sample sprite animation code for a rotating cursor. The code declares the required variables in the `Game` class.

```
/*Initialize sprite batch to hold sprite information*/
            SpriteBatch spriteBatch;
/*Texture of sprite that you want to render*/
            Texture2D textureImage;
```

```
      /*Position of Sprite*/

                  Vector2 position;

      /*The overall texture is a grid which holds sequential
images. The variable frameSize specifies each grid size .*/
                        Point frameSize;//each image size
/*currentFrame holds the current grid that is drawn*/

                        Point currentFrame;
/*sheetSize holds the total number of grid in the texture, both in
the Horizontal and in the vertical direction.*/

                        Point sheetSize;//overall texture size
                        float timeSinceLastFrame = 0.0f ;
/*You need to control the speed of animation. The variable
secondsPerFrame controls that.*/

                              float secondsPerFrame = 0.05f ;
```

The following code loads the texture in the `Game.LoadContent` method:

```
protected override void LoadContent()
{   //Load sprite texture

            textureImage = manager.Load<Texture2D>(texturePath);

}
```

In the `Game.Update` method, animate the sprite by setting the `currentFrame` to 0 (for example, the first grid of the image in the texture) and rendering the grid. You can check the variable `timeSinceLastFrame` for a predefined `timedelay`, which is in this case the value in the `secondsPerFrame`. When the time limit is crossed, move to the next grid by incrementing the value of the `currentFrame` and render the grid. This sequence continues producing an effect of an animating cursor. When you reach the end of the grid of a line, move to the next line. When all grids are covered, reset the counter to point to the first grid again. Figure 4-9 shows the original texture sheet for animating cursor.

Figure 4-9

The original texture sheet for animating cursor

Figure 4-10 shows how the texture sheet is divided with a grid of lines programmatically. By looking at the grid in Figure 4-9, you can infer that the size of the texture sheet can be set to Point (5, 6). As in Figure 4-9, the X-axis has five grids, and the Y-axis has six grids.

Figure 4-10

The texture sheet with grid

```
                    Protected override void Update(GameTime gametime)
                    {       //Increase current frame
                        timeSinceLastFrame +=gameTime;
                                if (timeSinceLastFrame >
secondsPerFrame)
                                    {
                                     timeSinceLastFrame -=
secondsPerFrame;
                                     /*Increment counter as delay is
                                  over .*/
                                    ++currentFrame.X;
/*Check if all the grids in the horizontal direction in a line are
covered*/
                                    if (currentFrame.X >= sheetSize.X)
                                    {
/*If yes, move the counter to the next line .*/
                                            currentFrame.X = 0;
                                           ++currentFrame.Y;
/*Check if all the lines are also covered*/
                                            if (currentFrame.Y >=
sheetSize.Y)
                                           /*If yes, set the counter to
first line again .*/

                                            currentFrame.Y = 0;
                                     }
                                }
```

The following Game.Draw method draws the sprite:

```
protected override void Draw(GameTime gametime)
{
/*Get mouse cursor position for drawing the sprite at
that point on the screen*/
MouseState currentMouseState = Mouse.GetState();
        position == new Vector2(currentMouseState.X,
        currentMouseState.Y);
/*Create a Rectangle object to specify the particular
area of texture you want to draw using the
information saved in the currentFrame variable */
    Rectangle rect = new Rectangle(currentFrame.X * frameSize.X,
currentFrame.Y * frameSize.Y, frameSize.X, frameSize.Y);
        spriteBatch.Begin();
        spriteBatch.Draw(textureImage, position,
        rect, Color.White);
        spriteBatch.End();
}
```

SCALING AND ROTATING MATRICES

As already discussed, the matrix structure in XNA contains 4 × 4 elements for the matrix values. You can use matrices for enlarging the objects in an animation and for rotating the objects around the axes. These are referred to as scaling and rotating matrices, respectively. You can also use a matrix for calculating the distance between two objects and changing the positions of both depending on the calculated distance, as in when you need to depict two cars colliding. Table 4-10 lists some of the methods available in the matrix structure for scaling and rotating purposes.

Table 4-10

Rotating and Scaling Methods of the Matrix Structure

Field	Description
CreateScale	This method creates a scaling matrix that can be used to scale an object.
CreateRotationX	This method creates a matrix that can be used to rotate a set of vertices around the X-axis. You can specify the rotation angle in radians.
CreateRotationY	This method creates a matrix that can be used to rotate a set of vertices around the Y-axis. You can specify the rotation angle in radians.
CreateRotationZ	This method creates a matrix that can be used to rotate a set of vertices around the Z-axis. You can specify the rotation angle in radians.

To double the size of an object, in order to look big, you use the scaling matrix. The following matrix generates a scaling matrix for all three axes. Figure 4-11 shows the scaling of an object using the scaling matrix.

```
Matrix result;
Matrix.CreateScale(2.0f, 2.0f, 2.0f, out result);
```

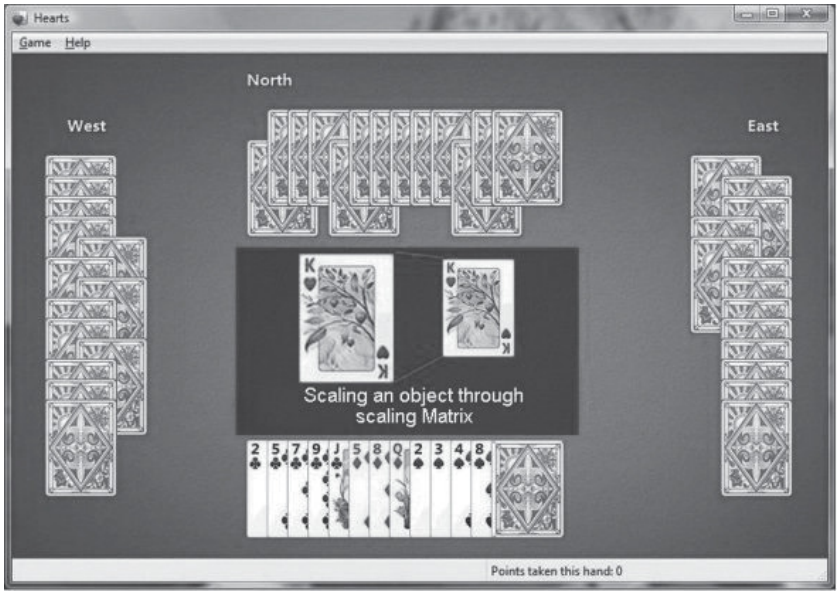


Figure 4-11

Scaling an object using the scaling matrix

The rotating matrix applies the sin and cos functions and signifies the individual rotation around the axes over an angle theta (Θ). The following code sample uses the `Matrix` class for rotation. Figure 4-12 shows the rotation of an object using the rotation matrix.

```
Matrix rotationmatrix;
//create rotation matrix around X axis
rotationmatrix = Matrix.CreateRotationX(2.0f);
//create rotation matrix around Y axis
rotationmatrix = Matrix.CreateRotationY(2.0f);

//create rotation matrix around Z axis
                rotationmatrix = Matrix.CreateRotationZ(2.0f);
```

The translation matrix calculates the distance between two objects and accordingly transforms or moves the two objects on the screen. The following code sample in the `Game.Update` method moves `object1` towards `object2`. Figure 4-13 shows the movement of an object using matrix translation. The code uses the `Normalize` method of the `Vector3` structure to create a unit vector from the direction of the `object1` in this case. The resultant vector is a unit vector pointing in the same direction as `object1`.

```
//Get the direction in which object1 has to move .
Vector3 direction = Object2Position - Object1Position;

/In 3D world, before using any vector we need to normalize it.
Mathematically
normalized vector of V = V vector / Magnitude of V
vector . It also can be treated as the UnitVector of V. */
```

```
Vector3.Normalize(direction);

//Now transfer object1 in that direction at a speed of 3 unit per
unit time.
Object1Position += direction * 3.0f * gameTime;

//Now set the world matrix for transformation.
Matrix world = Matrix.CreateTranslation(Object1Position);
```

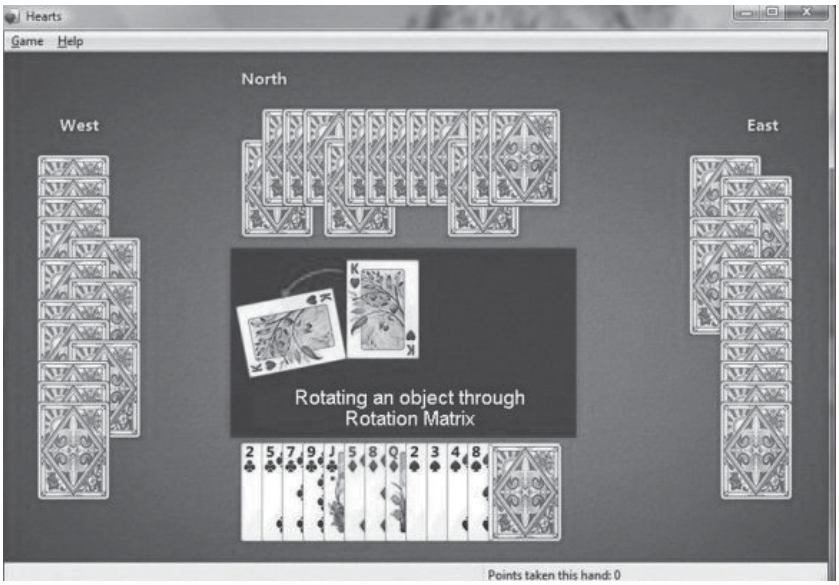


Figure 4-12

Rotating an object using the rotation matrix

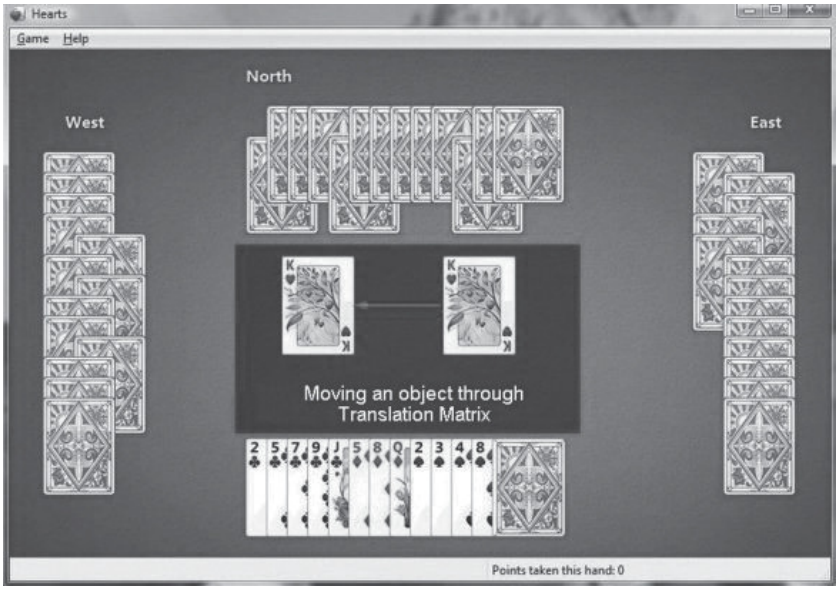


Figure 4-13

Movement of an object using the matrix translation

Creating the Feel of the Character

To ensure that you create a game that makes the player believe the characters within the game, you must create a feel of the characters through various techniques, such as lighting, shaders, projections, and so on.

In real life, when we look at a person, we take in much more than the person's physical characteristics. The way the shadow falls beside the person or the way the sunlight hits the person's hair can influence our perception of the person. For example, imagine that you are looking at a person who is angry and towering above you. If he blocks the sun and casts a shadow over you, you will naturally feel scared—a perception of fear that is increased by the shadow cast on you. It is therefore important to consider such elements when creating a game world that is as close to reality as possible.

The type of game that you are developing dictates whether you must consider creating the feel of the character in your game. For some kinds of games, you can avoid emphasizing the character's feel in the game. For example, in *Counter Strike*, the player is concerned only with killing the opponent and winning the game. Therefore, the feel of the character was not a consideration in that particular game.

In contrast, the setting for the video game shown in Figure 4-14 is creepy and dark. Therefore, there is ample use of various techniques to create a feel of the character and the game as such. For such games, you can create objects and characters easily, but the challenge lies in giving life, or in other words, creating the feel of the character.

Figure 4-14

Dark and creepy video game screen contrast

APPLYING FILTERS TO TEXTURES

CERTIFICATION READY
How can you manage the appearance of textures?
4.1

To make an image realistic, you use textures. A texture is a 2D image that is applied to a game object or character to define its surface. Sometimes creating a texture alone is not sufficient. You need to apply filters to your texture to create a realistic feel. For example, let us say that you have created an image of a rough, rocky terrain and the player character is moving on this terrain. When, in your game, you show a close-up of the character hitting the rocky ground and in pain, you will need the rocky terrain to appear as a coarse, grainy surface to complete the realism of pain on the character's face.

However, you might find that the XNA Framework displays a rather blurry, smooth terrain texture instead of a pixelated, rough look. This problem of textures losing their appearance is not due to anti-aliasing, but is actually due to the interpretation that the hardware makes of the textures.

TAKE NOTE*

Anti-aliasing is a software technique that diminishes the jagged edges to produce smooth edges. The computer screen uses pixels to draw objects. However, real-world objects are drawn using lines and curves; therefore, we always get jagged edges while drawing them on the screen. Anti-aliasing removes the edges by blurring them.

The XNA Framework helps us manage the appearance of the textures through the use of filters. A ***filter*** is a method or technique that enables you to change the appearance of textures on images.

You might need to keep applying different filtering methods within a game to create a feel of the character. At any point, you can use the `TextureFilter` field to see the current filtering method used in the game.

You need to understand the very basic input for a filter to work, which is the texture coordinates of the texture on which the filter is to be applied. The texture coordinates are represented as ***texels***, or texture element.

TAKE NOTE*

A texture consists of an array of pixels that are referred to as *texels*. A texel is the smallest unit of texture that the GPU can process. A texel gives the coordinates of the texture taking (0, 0) in the upper left and (w, h) in the lower right, where w is the width and h is the height of the texture image. The height and width are represented along a U and V axis, respectively, and are called *UV coordinates*. You can locate the format of the texel in the `SurfaceFormat` class enumeration of a `Texture2D` class object.

When you work with textures, you determine where in a particular space a texture will be applied, which is called ***texture mapping***. Moreover, you also need to determine whether you need to apply a filter on the texture or not. This decision depends on the texel information. When the texture is too large or too small for the given shape, then you need to use filters to either magnify or shrink the texture.

The `SamplerState` class in the XNA Framework does all the processing automatically through sampling. When you use the `SamplerState` class, you allow XNA to determine how to sample a texture or, in simple words, use a particular texture.

When you use the `SamplerState` class, if the texture is too big or too small for the image, appropriate filters are automatically applied on the texture. A magnification filter enlarges the texture by repeating the same texel values for multiple pixels, which can result in a blurred output.

A filter that shrinks the texture is more complex. This filter adds multiple texel values into one texel value. This can cause anti-aliasing or jagged edges on the final image. Therefore, a better way to shrink a texture is to use a ***mipmap***.

TAKE NOTE*

A mipmap is a sequence of textures that provides texture information in different levels. Each level of a mipmap provides a progressively lower resolution of the same image. A built-in formula ensures that each level is a smaller size than its previous level.

When the texture needs to be shrunk, the XNA Framework will automatically take the mipmap level that is closest to the required size. Size here refers to the image resolution.

In earlier versions of XNA, you needed to manually set the two filters separately. In XNA 4.0, however, the `SamplerState` class allows you to set the parameters for magnification and shrinkage of textures using the `Filter` property. We need to set the parameters for this `Filter` property in the `TextureFilter` class declaration part of the code. This `Filter` property includes the following values:

- Linear
- Point
- Anisotropic
- `LinearMipPoint`
- `PointMipLinear`

- `MinLinearMagPointMipLinear`
- `MinLinearMagPointMipPoint`
- `MinPointMagLinearMipLinear`
- `MinPointMagLinearMipPoint`

Each of these values directs the GPU to the type of filtering it needs to use when drawing the image. Basically, three types of texture filtering are used in the XNA Framework:

- ***Point filtering***: When choosing between two texels, the GPU can choose either of the two. The texture features of the chosen texel are used to draw the texture onto the pixel. This type of filtering causes abrupt changes in the texture of the final image. Point filtering is the most basic type of texture filtering.
- ***Linear filtering***: Using linear filtering, the GPU can calculate a texel value from the given two texels and use this resultant texel to create the final image. This filtering gives a smoother finish to the image compared to point texture filtering. However, linear filtering can cause a jagged edge finish and is not suitable for low distance textures.
- ***Anisotropic filtering***: Most texture filtering methods use a square filtering pattern that is applied in all directions in the same manner. A filtering pattern defines how the image texels are blurred. A square filtering pattern, or isotropic filtering, can cause aliasing or a jagged look at curves. In contrast, anistropic filtering uses a nonsquare filtering pattern that helps you avoid the jagged finish. This kind of filtering is particularly suited for detailing distant textures, such as mountains in the background or a long, winding road. However, anisotropic filtering requires a lot of memory and might cause the graphics performance to lag.

Table 4-11 presents the different filter property values available in the XNA Framework and their corresponding filtering method directions to the GPU.

Table 4-11

Filter Property Values with Corresponding Directions

Filter Value	Directions
Linear	Linear filtering
Point	Point filtering
Anistropic	Anisotropic filtering
`LinearMipPoint`	Magnify/Shrink: Linear filtering Between Mipmap levels: Point filtering
`PointMipLinear`	Magnify/Shrink: Point filtering Between Mipmap levels: Linear filtering
`MinLinearMagPointMipLinear`	Shrink: Linear filtering Magnify: Point filtering Between Mipmap levels: Linear filtering
`MinLinearMagPointMipPoint`	Shrink: Linear filtering Magnify: Point filtering Between Mipmap levels: Point filtering
`MinPointMagLinearMipLinear`	Shrink: Point filtering Magnify: Linear filtering Between Mipmap levels: Linear filtering
`MinPointMagLinearMipPoint`	Shrink: Point filtering Magnify: Linear Between Mipmap levels: Point filtering

When you use the SamplerStates class, you need to consider the default state and modify it if required for your game. The following code sample sets the property values for the SamplerStates class.

```
GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;
GraphicsDevice.SamplerState[0].Filter = TextureFilter.Linear ;
GraphicsDevice.SamplerState[0].AddressU = TextureAddressMode.Wrap;
GraphicsDevice.SamplerState[0].AddressV = TextureAddressMode.Wrap;
GraphicsDevice.SamplerState[0].AddressW = TextureAddressMode.Wrap;
GraphicsDevice.SamplerState[0].MaxAnisotrophy = 4 ;
```

TAKE NOTE*

SamplerState should be set properly before drawing a model or SpriteBatch.

The default states for the SamplerState class are:

1. It uses linear filtering.
2. It does not use mipmapping.
3. It sets the maximum anistrophy value to 4.

LIGHTING

CERTIFICATION READY
How is lighting handled in XNA 4.0?
4.1

Lighting is one of the most important and influential game design components that enhances the aesthetics of your game visuals. It has the power to make or break the visual theme and the atmosphere of a game. Without the lighting effect, the complete game environment would look flat and uninteresting. Lighting makes the visuals look scary or cozy. It augments the three-dimensional feel of the objects and creates composition and balance to lead the player's eyes around.

Lighting is an important tool to create the feel of a character, because certain perceptions associated with a character can be formed through the clever use of lighting. For example, most villain characters in games are featured to remain or move through shadows to highlight their dark nature. Games such as *DOOM 3*, *Resident Evil*, and *Silent Hill* are set in a gloomy, dark world. The use of reduced light, shadows, and dull color tones help to create a creepy feeling in the player. On the other hand, fantasy-based games—such as *Super Mario*, *Aladin*, and *Mystery Island*—are bright and colorful with the use of light to create an adventurous environment.

A few aspects of lighting can be addressed during character design itself. For example, a villain character might wear dark-colored clothing for ease in blending with the shadows. Lighting as a consideration does not end with character design. In fact, it moves with the character, quite literally. When a character moves, the effect of light falling on him or her must also be captured realistically in the game. For example, if the villain character moves from the shadows into light, the character must immediately reflect the change in terms of becoming more visible.

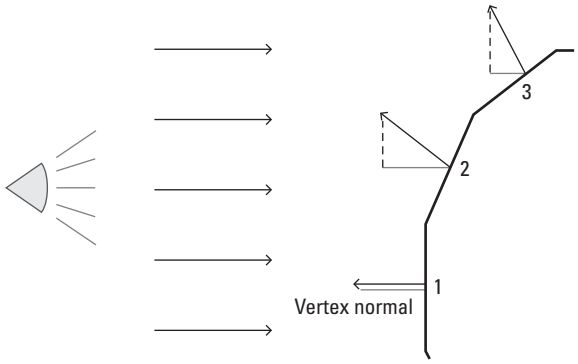
In *DOOM*, there are mutated evil characters in every corner of the spaceship, but something has gone wrong and the light of the entire spaceship is flickering. Some of the passages are completely dark and the player character needs to use his or her flashlight to see where the character is going. At a point when the player character hears something and directs the light to it, the character sees an evil character about to pounce. This is an example in which the light element plays a vital role in creating a fear factor and building the suspense in the game for the player.

Providing proper lighting in games involves calculating the resulting color of the surrounding objects or characters when light falls on them. Today, game engines provide predefined properties and functions that you can use to create lighting in games. The XNA Framework uses a built-in formula that helps you predict the amount of directional light falling on an object. This helps you to then define the parameters of the resulting colors to ensure a smooth effect of the light on the object.

To understand this technique, let us consider a triangle. Figure 4-15 displays an illustrated example of this technique.

Figure 4-15

Illustration of an example showing the working of lighting in XNA



In Figure 4-15, a directional light, such as sunlight, falls on one side of the triangle. Note that the side facing the sun is brighter or, in graphical terms, must be seen more than the other side. To represent this in terms of the image color, the XNA Framework requires two inputs—the direction of the light and the "normal" at each vertex. Vertex normal means a `Vector3` component that is perpendicular to the vertex. Each vertex is associated with vertex normal information. In XNA, it is `Vector3` object.

The XNA Framework uses its built-in formula to calculate this amount of light automatically, saving you a lot of manual calculations. You simply need to provide the XNA Framework the angle that the reflected ray will make with the light ray, or angle of incidence.

The XNA engine projects the amount of light to be reflected as a vector. This vector output is used as an input parameter for rendering color. The XNA Framework allows you to share a vector common among two vertices to ensure a smooth effect. You can therefore avoid edged lighting or sharp color contrasts.

The XNA Framework considers any surface to be made up of triangles, which means that you can use the built-in formula for calculating the amount of light not just on triangles, but also on a character's face. This enables you to give a smooth, realistic feel to your character.

For 3D characters and objects, the XNA framework uses ***per-pixel lighting***. In this technique, the light source is a point in 3D space and shines light in every direction. Every object and character in the 3D space is lit by a certain amount of light. The XNA Framework calculates this amount of light by multiplying the vectors of the normal and the direction of light for each pixel of the image. The result is the amount of light reflected toward the camera.

XNA Framework provides an built-in `DirectionalLight` class to produce the lighting effect. Directional light has a direction but no position. It does not have a falloff value. Sunlight or moonlight is considered as directional light. However, in a 3D world, point light and spot light can also be used.

Point light has a position but no particular direction because it can emit light in all directions. Sometimes, it is also referred to as *Omni light*. It is always associated with a falloff value. The intensity of light decreases as it moves away from the source. A candle can be considered as a point light. Spot light has a position and direction. It spreads its light in a cone-shaped area along a specified direction. It is also associated with a falloff value. It has two falloff values: one is a linear falloff that determines the amount of light decreased in the direction of light, and two is a falloff value that decides the amount of light at the edges of the cone. Flash light can be considered as spot light. The following shows a sample spot light and point light class declaration.

```
class  pointLight
{
       Vector3 position ;
       Color  color;

       float   falloff ; //user defined variable that contains
the fall off value.
}

class  spotlight
{
       Vector3 position;
       Vector3 direction;
       Color  color ;
       float  coneAngle;
       float  linearFalloff; //user defined variable that
contains the linear fall off value.

       float  penumbraFalloff ; //user defined variable that
contains the fall off value.

}
```

Figure 4-16 shows a character's face and arm lit up using per-pixel lighting.

Figure 4-16

An example of per-pixel lighting in XNA

Usually, you define a light source and a character or object. Often, you end up replicating a lot. To simplify your 3D lighting work in XNA, you can use `BasicEffects`, a simple effect available with Models in XNA 4.0. This effect can support up to three directional lights and is the default effect in use when you render your 3D models or representations for characters, sets, and props within the game. You can use this feature if you want a number of characters

and objects to have the same lighting effect from a single light source. For example, you can show all the characters and objects under the sun in the same open terrain.

SHADERS

CERTIFICATION READY
How will you create shadows of your game objects in XNA 4.0?
4.1

Light and shadow are two sides of a coin. Together, these elements breathe life into an image. Although lighting is the tool that allows you to re-create the perfect lighting effect in your game, shaders allow you to effect changes to the characteristics of a surface to complete the illusion created through lighting.

A ***shader*** is the surface property of an object or character. It includes characteristics such as color, shine, reflection, refraction, roughness, and so on of a particular surface. A shader program is executed by the graphics processing unit (GPU) for each vertex or pixel to effect specific changes to the surface property.

Shaders offer flexibility in terms of graphics processing. You can use the ***High Level Shading Language (HLSL)***, which is a proprietary shading language developed by Microsoft for use with Direct3D for shader development. HLSL is similar to C. Like in C programming, you can also declare variables, functions, data types, and use `if/else/for/do/while` loops using the HLSL. The GPU runs shaders on the graphics card of the end user's hardware.

MORE INFORMATION

For more information on HLSL syntax and supported data types and functions, refer to the section, "Reference for HLSL," in the MSDN Library.

You can create shaders using the Visual Studio Editor in an effect file. An effect file has an extension .fx and is a text file. It contains three main sections: one is for variable declarations, the second is for techniques and passes, and the third is for functions.

Techniques and passes contain the definition of vertex and pixel shader functions that defines how an object is to be rendered. An effect file can contain one or more techniques. A technique can have more than one pass. The following is the sample technique in an effect file.

```
technique technique1
{
      pass P1
      {
            VertexShader = compile vs_2_0 VS();
            PixelShader = compile ps_2_0 PS();
      }

}
```

You can use shaders in all 3D drawings created in the XNA Framework. The XNA Framework supports only HLSL.

There are two kinds of shaders in XNA, vertex shader and pixel shader.

The ***vertex shader*** takes the 3D position of the vertex, multiplies it with a matrix, and then transforms it into 2D screen coordinates. This feature speeds up the game, as less time is taken for processing 2D coordinates by the GPU. You can use the vertex shader to make a character fatter or thinner, shorter or taller, or make any other modifications that deal with the position of the vertices.

You can control the output that a vertex shader will present to the pixel shader. For this, you simply define the data in the structure of the vertex shader and then make the shader return the value of the data as output. The following shader code sample illustrates this:

```
      float4x4 World;//World matrix of object
      float4x4 View ;//View matrix of Camera
      float4x4 Projection;//Projection Matrix of Camera
      /*Declare a custom vertex structure. This structure
signifies how vertex data will be read from the graphics
pipeline. */
      struct VS_IN
      {
          float3 Position : POSITION0; /*holds vertex
position */
          float3 Normal : NORMAL; /* holds vertex normal
information */
          float2 TexCoord : TEXCOORD0; /* holds texture
coordinate that tells how to apply texture on an object */
          float3 Tangent : TANGENT; /* holds the tangent
information of vertex */

      };
      //Define custom Vertex output method.This one is same as
Vertex input structure .This structure holds the processed ver-
tex information from vertex shader stage .
      struct VS_OUT
      {
          float4 Position : POSITION0;
          float2 TexCoord : TEXCOORD0;
      };
      //Declare texture
      texture g_txScene;
      //Define Sampler state of texture
      /*This holds the information how to apply texture to
object . You can define different MipMap filter for texture and
also texture addressing mode . */

      sampler g_Texture = sampler_state
      {
          Texture = (g_txScene);
          MAGFILTER = LINEAR;
          MINFILTER = LINEAR;
          MIPFILTER = LINEAR;
          AddressU = Wrap;
          AddressV = Wrap;
      };
      /*Vertex shader to process the vertex information.*/
      VS_OUT VS_BasicMesh(VS_IN input)
      {
         VS_OUT output;
         //Transform from object space to world space
         float4 worldPosition = mul(float4(input.Position,1),
World);
      /*Transform vertices from world space to View Space by
multiplying it with View Matrix*/
         float4 viewPosition = mul(worldPosition, View);
      /*Transform vertices from View space to Screen Space by
multiplying it with Projection Matrix*/

         output.Position =  mul(viewPosition, Projection);
```

```
        /*Copy the texture information as it is */
        output.TexCoord = input.TexCoord;

              return output;
        }
```

The ***pixel shader*** draws each pixel of the screen using the coordinates it receives from the vertex shader. The pixel shader calculates only the color for each pixel. The following code sample illustrates a pixel shader. The pixel shader takes the coordinates from the vertex shader and returns the color of the coordinates. In the sample code, you see that `COLOR0` is added after the function name. It is a semantic, telling you that the output contains only `COLOR` information.

```
float4 PS_BasicMesh(VS_OUT input):COLOR0
{
   //Sampling the texture
   float4  output = tex2D( g_Texture, input.TexCoord );

    return output;
}
```

Finally, define the vertex shader and pixel shader functions inside a technique. This tells us that the `technique1` will use the `VS_BasicMesh()` function as the vertex shader, the `PS_BasicMesh()` function as the pixel shader, and the shader requires shader model 2.0 or higher.

```
technique Technique1
{
    pass Pass1
    {
         VertexShader = compile vs_2_0 VS_BasicMesh ();
         PixelShader = compile ps_2_0 PS_BasicMesh ();
    }
}
```

Let us now understand how the XNA Framework allows you to implement shaders. Loading and using a shader in the XNA Framework consists of a few simple steps:

1. You need to first make your shader. For this, right-click the *Content Node* and click *Add* > *New Item*. In the Add New Item dialog box that opens, choose the *Effect File* from the list of templates (see Figure 4-17). An effect file is opened with the default name Effect1. Type the shader code in the Effect1.fx file.
2. Give an asset name to your shader file. Generally, the asset name is the same as the .fx file name. However, you can edit the asset name as you want. Figure 4-18 shows the Effect1.fx file and the Properties page listing the Effect1.fx file properties in the Visual Studio Development Environment.
3. To load the effect file in XNA, you next need to use an instance of the `Effects` class. The `Effects` class in the `Microsoft.Xna.Framework.Graphics` loads and compiles your shaders.
4. Then, you need to use the `Content` property to initialize the shader. The following code loads the shader using the instance of an effect class.

   ```
   Effect effect = Content.Load<Effect>("Effect1");
   //Effect1 is the asset name.
   ```

5. You must select the technique that you want to use. A technique represents the function of a particular effects file. For example, your shader can use a technique called *Sunlight* to make the image appear brighter on the screen. The following code loads the technique `technique1` from the Effect1.fx file.

Figure 4-17

The Add New Item dialog box listing the shader file template

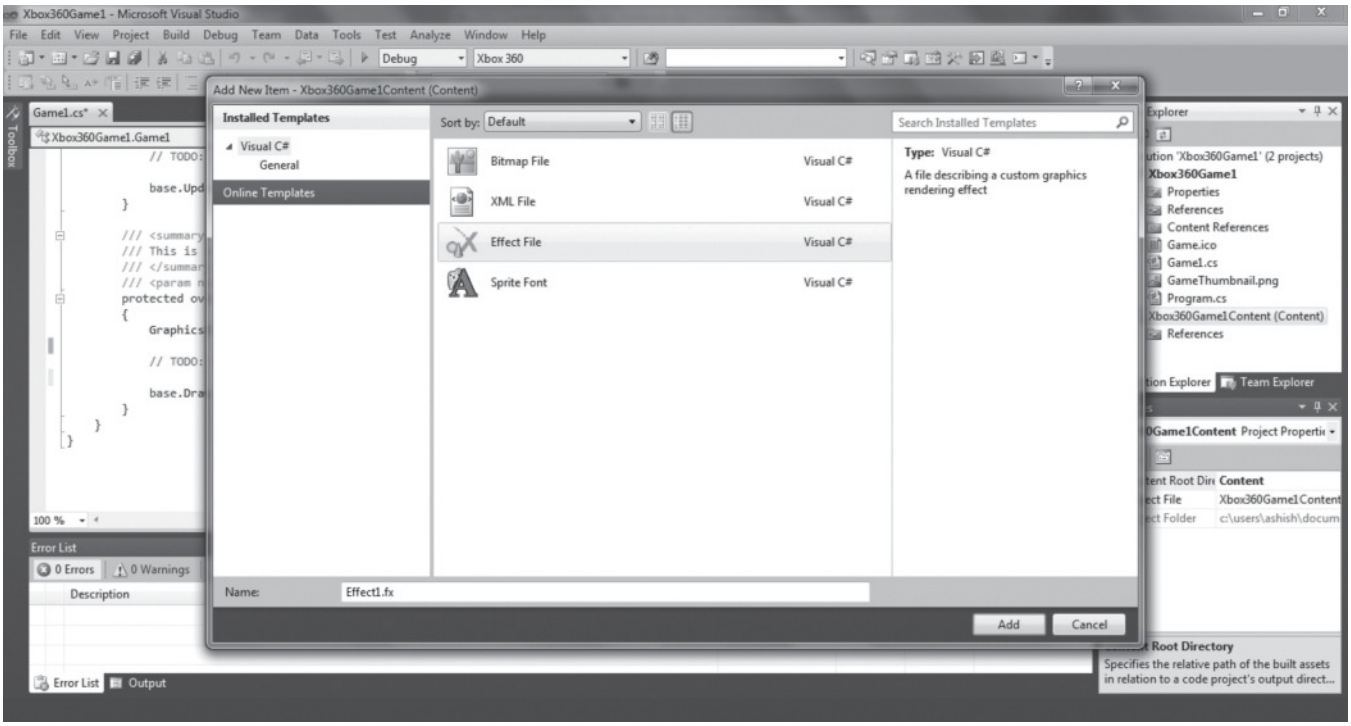
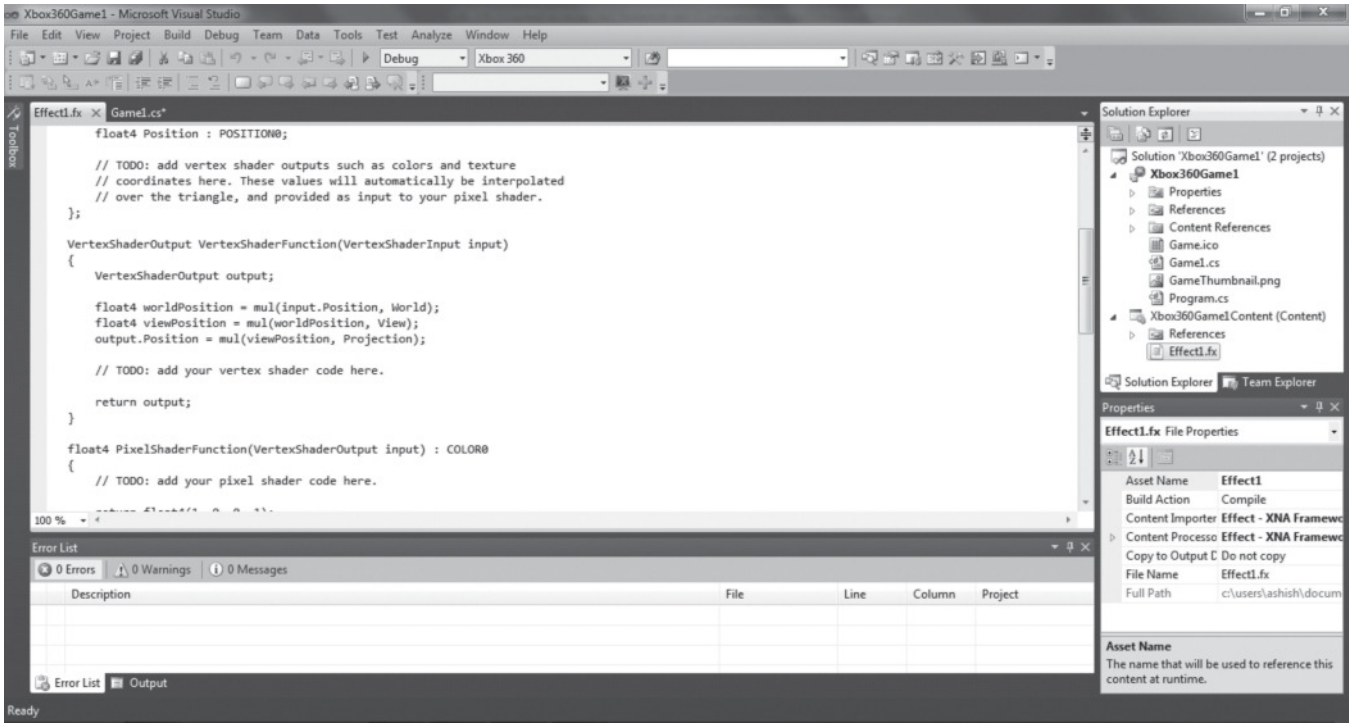


Figure 4-18

The Effect file with properties



TAKE NOTE*

When you compile your game code by pressing F6 in Visual Studio, your shader file will also compile automatically.

```
effect.CurrentTechnique =
    effect.Techniques["technique1"];
```

6. You then must pass the parameters you want to set in your shader using the `EffectParameter` object. These parameters capture the output of the shader.
7. Finally, you need to render or draw the object or character using a loop.

The following steps along with the code discuss how to draw a model in the XNA Framework using the shader (Effect1.fx) created above.

USE SHADERS

GET READY. To draw a model using the shader, do the following:

1. Declare a Model object and an Effect object in your Game class.

```
Model currentModel;
Effect currentEffect;
```

2. In the `Game.LoadContent` method, load the model and the shader. The code uses the car model that was discussed in the "Form Objects" section.

```
currentModel = Content.Load<Model>("Car.fbx");
currentEffect = Content.Load<Effect>("Effect1.fx");
```

3. In the `Game.Draw` method, draw the model. Before drawing the model, set the parameters that were initially declared in the shader file. The parameters include "world," "view," and "projection." Note that these are user-defined variables of type matrix. In HLSL, matrices are defined as a float 4 × 4 type and it is equivalent to XNA matrix.

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

    /*To draw a model using the above shader , we need to acess
ModelMeshPart of each ModelMesh of currentModel which can be done
like this .*/
        ModelMesh mesh = currentModel.Meshes[0];
        ModelMeshPart meshPart = mesh.MeshParts[0];

        //Initialize View matrix for Drawing the object
        Matrix viewMatrix = Matrix.CreateLookAt(new
Vector3(0.0f, 20.0f, 20.0f),
Vector3.Zero,Vector3.Up);

    //Initialize projection matrix for Drawing Object
        Matrix projectionMatrix =
Matrix.CreatePerspectiveFieldOfView((float)Math.PI / 4.0f,
1.33f, 0.01f, 1000.0f);

    /*Now set the parameters of shader. Set the world matrix to
Identity as we want to see the car at Origin*/
        currentEffect.Parameters["World"].SetValue(Matrix.Identity);
    /*Set the view matrix .*/
      currentEffect.Parameters["View"].SetValue(viewMatrix);
    /*Set the Projection matrix .*/
           currentEffect.Parameters["Projection"].SetValue
      (projectionMatrix);

    /*Now set the current Technique of Shader . Each Technique
defines a Vertex shader and Pixel shader.Effect1.fx has a technique
named technique1*/
```

```
                currentEffect.CurrentTechnique =
            currentEffect.Techniques["Technique1"];

        /*After setting the technique, we need to iterate through all
pass available in that technique . Effect.fx shader file has only
one pass.*/

                for (int i = 0; i <
            currentEffect.CurrentTechnique.Passes.Count; i++)
            {
                /*EffectPass.Apply will update the device to
                    begin using the state information defined
in the current pass*/
            currentEffect.CurrentTechnique.Passes[i].Apply();

            /*We need to set the Culling mode so that unnecessary
         triangles will not be processed. CullMode is defined as a
         member of RasterizeState in XNA which defines
the current render state of Device*/
            RasterizerState state = new RasterizerState();
            state.CullMode = CullMode.CullCounterClockwiseFace;
            graphics.GraphicsDevice.RasterizerState = state;

            /* Draw the car model using the DrawIndexedPrimitives
         method using the TriangleList primitive. Because, when you
         export any model created in a 3D software using the .fbx or
         .x file format, the file contains the information of the model
         in terms of the primitive shape triangle. Therefore you can
         use the primitive methods of the graphics object to render the
         model */

            graphics.GraphicsDevice.DrawIndexedPrimitives(
           PrimitiveType.TriangleList, 0, 0,
            meshPart.NumVertices, meshPart.StartIndex,
meshPart.PrimitiveCount);
            }
                    }
```

When you use lighting, you get the parameters for the amount of light that will define the other surface properties, such as shine, color, and so on. You can use these parameters to draw the object or character. When you combine lighting with shaders, you can input the parameters from lighting to create the right effect of light on the image. The right kind of light and surface properties reflecting the effect of light will bring the character of your game to life.

Generally, surface is always associated with material property. Light is always reflected or absorbed according to the surface material.

Depending on the surface material, there are two types of lighting calculations:

- Diffuse
- Specular (blinn and phong)

The following code calculates the diffuse light:

```
     /* Formula for calculating diffuse light . It performs a vector3
dot product to calculate light */
     diffuseLight = dot(normal,lightdirection);
```

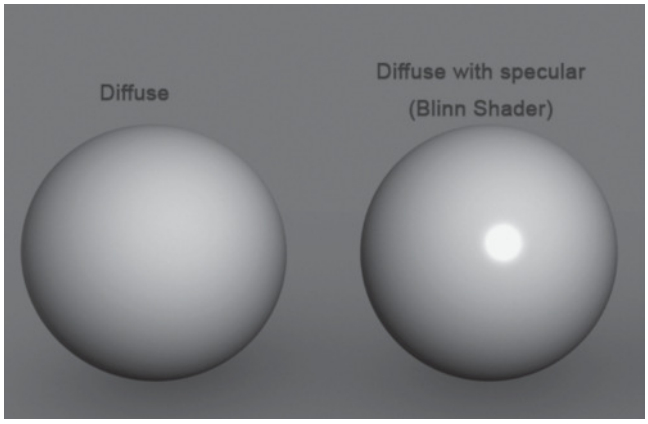
The following code calculates the specular light:

```
      //Calculate camera direction
      float4 cameradirection = worldPOsitionOfVertex -
CameraPOsition;
      //Calculate half vectore
      float4 vhalf = normalize(light + cameradirection);
      //Calculate secular light .
      float specular = dot(normal, vhalf);
```

Figure 4-19 shows an image with the diffuse and specular blinn lighting effects.

Figure 4-19

Examples of different lighting effects



Shaders are important when you use shadows for your images because the shadow quality depends on the object. For example, a transparent object must have a lighter shadow than an opaque object.

There are simple steps to add shadows:

1. You need to first draw all the objects and the character in black color around the same light source onto a texture. This image is called a *castor map*.
2. You then use a shader to convert the castor map into a shadow's image.
3. Finally, you need to blend the shadow's image with the ground or texture on which the objects and characters lie.

To render the scene, you need the view and projection matrices of camera. Similarly for shadow, you need the view and projection matrices for the light object. So, first, create the view and projection matrices of light. Next, render the scene with that light view and projection matrices. Then perform the shadow calculation. View and projection matrix can be created for any object in XNA. For example, the view matrix of spot light can be created as shown in the following:

```
Matrix viewMatrix =
    Matrix.CreateLookAt(spotLightPosition,
      spotLightDirection,
      Vector3.Up);
Matrix projectionMatrix =
Matrix.CreatePerspectiveFieldOfView(coneAngleofSpotLight ,
1.33f, 0.01f, 1000.0f);
```

CERTIFICATION READY
How can you implement projections in XNA?
4.1

PROJECTIONS

Creating the feel of a character also includes the world around the character. This is because whatever the character does or says is in response to the occurrences around the character. Realism to the character's experiences is created only if there is realism in the world around

the character. For example, to make the player believe that he or she can actually move around the game world in the same ways as the player does in the real world, we need to ensure that the player character's view changes along with the character's movement as it does in the real world. That is, if you move close to a flower in the game world, the flower must appear larger in size.

In a game, you need to use projections to effect scaling and view changes to the character or an object's environment. You can also use projections to throw blood splats on a wall when a player character standing close to the wall is hit (see Figure 4-20). This brings in realism because in the real world, in such a situation, the character's blood will splatter both on the wall and the ground.

Figure 4-20

An example showing the projection of blood splatter

To understand projections, let us recall the world space, view space, and Projection space that we discussed earlier in this lesson.

- **World space**: Your gameplay takes place in this 3D space. It can use the entire screen space available.
- **View space**: This 3D space defines the proposed visible objects and how those objects are visible from the game camera. View space performs culling and clipping operations on an object.
- **Projection space:** GPU uses this 2D space for final rendering or drawing of the images on screen. This space starts from (−1, −1) in the lower left and ends at (1, 1) or upper right of the screen. It is this space that is visible to the player. Therefore, it is important for you to ensure that all final images fall within this space. For this, the 3D coordinates of the game models should be transformed into 2D screen coordinates.

In the XNA Framework, we can use projections through `SpriteBatch` or through creation of a projection or scaling matrix. Both techniques use the three kinds of space that are involved in game development.

Although 3D images generally require a projection matrix for transformational effects, 2D images can be transformed easily using `SpriteBatch`.

SpriteBatch

The `SpriteBatch` class helps you to draw a group of sprites having the same settings. The `SpriteBatch` class is used primarily for rendering 2D images. It handles the projection space automatically for you. To do this, `SpriteBatch` uses a projection matrix internally that converts the client space to projection space. Client space for `SpriteBatch` starts in the upper-left corner of the Viewport or the 3D area in which the game developer is working.

In SpriteBatch, one unit of client space is equal to one pixel in Viewport. SpriteBatch scales the sizes and positions of every sprite drawn.

You can move an object within the world space using SpriteBatch.Draw. This implies that a 2D character or object movement and the associated view and scaling changes are easily rendered when you use SpriteBatch. This is because the transformations can be simply given in SpriteBatch and the changes to the images are effected before the images are drawn. This makes your job easier and faster.

PROJECTION MATRIX

As already discussed, this matrix converts the world space to projection space in order to help the GPU draw the image. If you use 3D images and BasicEffects, you need to provide a projection matrix to handle the transformations and other scaling effects. This is because unlike SpriteBatch, BasicEffects does not have any built-in functionality.

There are two types of projection matrix, orthographic and perspective. In general, when you look at a scene, the distant objects appear smaller than the objects that are close by. The perspective projection maintains this realism. This projection shows distant objects as smaller and it gives the feeling of depth. However, an orthographic projection ignores this realistic effect to provide accurate measurements. That is, in an orthographic projection, both the distant objects and closer objects look the same size.

SpriteBatch uses orthographic projection for screen space transformation as depth is not required. Models in 3D images are rendered by using perspective projection as depth is important.

The following code sample uses the projection matrix:

```
Matrix.CreateOrthographicOffCenter (Left, Right, Bottom, Top,
zNearPlane, zFarPlane)

Perspective Projection =
Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
GraphicsDevice.Viewport.AspectRatio, 1.0f, 1000f);

Orthographic Projection =
Matrix.CreateOrthographicOffCenter(0,
GraphicsDevice.Viewport.Width,
GraphicsDevice.Viewport.Height, 0, 1.0f, 1000.0f);
```

Similar to a projection matrix, you can also create and use a world matrix to make the object move within the world space. You can also create a view matrix to use as a camera and pan the view around the object or character. This feature can be useful if you want the character to follow another character.

Depending on the type of game you are developing, sometimes you might have to blend the two kinds of techniques. You can harness the optimization of SpriteBatch with the customization of projection matrices to create the right feel of a character.

One example of using a blended approach is when your game does not follow the set parameters defined by SpriteBatch. SpriteBatch assumes that the entire client space is used by the world space. SpriteBatch takes a horizontal direction on the Y-axis; therefore, if your game has a different orientation of the coordinate plane, you need to change the dimensions accordingly by performing the following procedure:

1. Create an additional matrix to convert from world space to client space.
2. Insert this matrix between the view and projection matrices.
3. Then multiply with more information about the screen that is presented in View and pass the output to SpriteBatch.

The following code statement is an example of an additional matrix:

```
Matrix.CreateScale(viewport.Width /1.5f,
viewport.height /1.5f, 1f);

/*where you are basically scaling up the image and
the image dimensions are 1.5 by 1.5 units.*/
```

You can also use `SpriteBatch` with custom vertex shaders. This gives you the flexibility of customizing the output and then using this customization with multiple sprites, which in turn reduces a lot of time and effort. You should also factor in any perspective distortion that can happen, especially when you work with 3D images.

GENERATING OBJECTS WITH USER-INDEXED PRIMITIVES

CERTIFICATION READY
What are the different primitive types available in XNA 4.0 ?
4.1

Throughout the game, based on changes to the player character's environment, you need to update the feel of the character. This means that you might require at some point in the game to generate objects using a set of coordinates and properties that you define that will direct the GPU to draw custom made objects.

The XNA Framework allows you to create custom-made objects using the primitive methods of the `GraphicsDevice` class such as the `DrawUserIndexedPrimitives`. Using this method, you can direct the GPU on how you want it to interpret and, therefore, draw vertices stored in an array. The resultant 3D shape is called a *primitive*. You can use the primitive type enumeration discussed in the "Form Objects" section to draw the required primitive geometry.

To render or draw primitives, you must create an effect and a transformation matrix. For this, you need to use a `BasicEffect` instance. Using the `DrawUserIndexedPrimitives` method of the Graphics device, you can create simple points, and lines, to triangles, textured quadrilaterals, and the most complex 3D shapes possible.

The following steps illustrate how to draw lines using the `DrawUserIndexedPrimitives` method.

DRAW A LINE

GET READY. To draw lines using the `DrawUserIndexedPrimitives` method, do the following:

1. Declare the following variable in your `Game` class. The `VertexPositionColor` used in the code is a structure to hold position and color information of a vertex. The code declares an array of the `VertexPositionColor` structure, as you need two vertices to draw a line.

```
        /*declare vertex properties */
      VertexPositionColor[] vertices;
      /*declare an array of uint to hold indices
.Indices are the information used to decide how vertices
will be connected to each other */

      uint[] lineListIndices;

      /* The BasicEffect is a built-in class in XNA
which is used to contain basic rendering effect.*/
      BasicEffect basicEffect ;
```

2. Create a method called `InitializeVertices` in the `Game` class to define the properties of the vertices.

```
      public void Initializevertices ()
      {
      /*Defines the vertex properties such as the position and
color information associated with each Vertices. We need this
information to draw and update the line */
```

```
        vertices = new VertexPositionColor[2];
        /* Position and color info for 1st vertex .*/
        vertices[0].Position = new Vector3(-0.5f, -0.5f, 0f);
        vertices[0].Color = Color.Red;
        /* Position and color info for 2nd vertex .*/
        vertices[1].Position = new Vector3(0, 0.5f, 0f);
        vertices[1].Color = Color.Green;

        /*Initialize indices */
        lineListIndices = new short[2] { 0, 1};

        /*Now define your BasicEffect object as it's going to be
used while drawing the triangle.*/

        basicEffect = new BasicEffect(GraphicsDevice);
        }
```

3. Override the `LoadContent` method by calling the `InitializeVertices` method.

```
    protected override void LoadContent()
    {
          Initializevertices();
    }
```

4. Override the `Game.Draw` method to draw the line as shown in the following:

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

    /*Set the culling state for Device to CullMode.None so that
line will always be visible*/
    RasterizerState state = new RasterizerState();
    state.CullMode = CullMode.None;
    GraphicsDevice.RasterizerState = state;

    //Set the basic effect parameter to show the color of
    each vertex.
    basicEffect.VertexColorEnabled = true;

    /*Now iterate through all passes of currentTechnique
    of basic effect and draw the line */
    foreach (EffectPass pass in
    basicEffect.CurrentTechnique.Passes)
    {
                    pass.Apply();
                    /* Draw the line using
    DrawUserIndexedPrimitives function of the GraphicsDevice*/

    GraphicsDevice.DrawUserIndexedPrimitives<VertexPosition
    Color>(PrimitiveType.LineList, vertices, 0, 2, lineList
    Indices, 0, 1, VertexPositionColor.VertexDeclaration);

      }

    }
```

You can also use a primitive type of `LineStrip` to draw lines. The difference is that with `LineStrip`, you are actually drawing the line as a series of connected lines.

DRAWING TEXTURED QUADRILATERALS

Drawing points, lines, and even 3D triangles might seem easy. Next in the difficulty level is drawing a textured quadrilateral. A quadrilateral can be split into triangles. Therefore, once you create triangles and textured quadrilaterals, you can draw a custom user-indexed object. To draw a textured quadrilateral, you need to:

1. Create a function to calculate the four corners of the quadrilateral. In this, you provide the origin, height, width, and the surface that needs to face the player. This function needs to call another function, which applies the texture on the quadrilateral.
2. To create the function that will apply the texture on the quadrilateral, you must provide the textel coordinates for the function to use. You must also provide the vertex positions to the function.
3. You need to specify the order in which the vertices of the quadrilateral are to be drawn using a primitive type of `TriangleList`.
4. To finally draw the quadrilateral, in the game `Initialize` method, create the `Quadrilateral` object.
5. In the `LoadContentGraphics` method, create a `BasicEffect` object for the texture you want to apply on the quadrilateral. Here, you also need to create the vertex type for the quadrilateral.
6. Finally, in the `Draw` method, in every pass, use `DrawUserIndexedPrimitives` to draw the quadrilateral.

CREATING CUSTOM VERTEX

A custom vertex allows you to draw 3D user-defined objects with your own choice of color and lighting effects. This gives you the flexibility to create your own custom data type, define its properties and effects, and then use it to draw your own 3D objects.

When you create a custom vertex, you must define:

- Properties—such as position, color, and normal—for each vertex
- A public method called `SizeInBytes`
- An array of type `VertexElement` to represent each vertex property

Each vertex can have properties such as position, color, normal, tangent, binormal, and texture coordinate. Using `VertexElement`, you can define the properties for vertices. Doing so enables you to access values of the vertex properties and process them using Shader. For example, you can use the normal property for calculating light, or the texture coordinates for applying texture, and so on.

Once you create your custom vertex, you can use `BasicEffects` to set the effect parameters for your custom vertex. You also need to initialize the view and projection matrices so that the 3D object can be drawn and seen on the screen properly.

To use a custom vertex for user-indexed primitives, simply initialize the array of user indexes with the custom vertex. This then passes on the properties of the custom vertex onto the user index primitive that is finally drawn on the screen.

■ Designing Physics-Based Animations

THE BOTTOM LINE

Physics-based animations help you mimic real life more closely as compared to traditional keyframe animations. Physics-based animations work by solving mathematical equations using iterative methods. The process considers some predefined constraints for conservation of mass, energy, and angular momentum of the object.

CERTIFICATION READY
What are physics-based animations?
4.3

Game designers have been using keyframe animations to animate characters and objects in games for ages and are still using them, albeit with improved technology. These animations, also called *scripted animations*, are ideal for predetermined sequences that form a part of the storyline. To create these animations, you need skilled artists to script a detailed storyboard and create a number of keyframes or line drawings. Artists then draw the frames that will go in between the keyframes. The number of keyframes and in-between frames depends on the level of realism required. Therefore, the effort scales up in proportion to complexity.

To run this type of animation, the production team needs to stay highly organized and render the frames systematically, leaving no scope for error. Because the animation happens by interpolating keyframes, the result might show invalid configurations (pass-through objects) and unnatural or jerky motion.

Game designers are always interested in developing game characters that can simulate a real human in terms of visual appearance, motor skills, and (ultimately) reasoning and intelligence. This constant desire and need led to the advent of physics-based animations.

Physics-based animation, in contrast to traditional keyframe animation, mimics real life more closely; the realism results from physics modeling. The resultant animation is interactive; that is, the player can change the way the object or character behaves by choosing the appropriate options. This capability, when increased multifold in magnitude, becomes artificial intelligence (AI).

Despite the sophisticated results that can be obtained from physics-based animations, these animations are simpler to program than traditional keyframe animations. This is because most of the processing is taken care of by the physics engine that runs on the laws of physics. Let us understand these technologies and their contribution to game development.

Understanding Physics Simulation

Physics simulation is the technique of incorporating the laws of physics into the game engine to more realistically simulate the movement of game objects and characters that are free to move in a 3D space. It involves physics calculations taking random values.

Physics-based animation involves applying the laws of physics to a simulation. The result is called a *physics simulation*. Like any simulation, physics simulation requires us to develop a model that represents the key features of the real system or process. The simulation represents how the system or process will behave or perform over a period of time.

In the gaming world, this model is the 3D model of the object or character that needs to perform specific actions in the game. Figure 4-21 shows a 3D game model.

Physics simulation helps game developers provide realistic motions to their objects and characters. To better understand how the laws of physics are employed into the game motion, let us look at two examples:

- **Example 1**: In your shooting game, imagine a scene in which the game player has to shoot an apple hanging on a tree branch to complete a level. If the player aims at this target and hits it, it is expected that the apple will fall to the ground, in line with the laws of gravity.
- **Example 2**: Imagine a basketball match in your sports game in which a game character bounces a ball before throwing the ball into the basket. In this case, the ball is expected to bounce back immediately upon impact with the ground.

From the previous examples, it is clear that physics simulation plays an important part in providing the overall game experience. It helps you to develop games that provide realistic simulations of object-player interaction and animation of objects based on concepts of physics, such as applied forces and environmental resistance.

Figure 4-21

An example of a 3D game model

PHYSICS CONCEPTS

Most of the physics simulations applied in game animation are based on the three Newton's laws of motion. The following list discusses some of the physics concepts based on these laws of motion:

- **Linear motion:** Linear motion is movement of an object along a straight line with constant acceleration or variable acceleration. An example of linear motion in games is a game character running in a race along a straight track.
- **Angular motion:** Angular motion is movement of an object along the circumference of a circle with or without a constant angular rate or speed of rotation. An example of angular motion in games is a car going around a curve on a racetrack in a racing game.
- **Force:** Force is any power that, when exerted upon an object, causes the object to undergo a change with respect to its movement, direction, or geometrical structure. An example of force in games is tug-of-war in a sports game in which two teams exert force by pulling at opposite ends of a rope with the aim of dragging the opponent team over a central line.
- **Inertia:** Inertia is the resistance of an object to any change in its state of motion. An example of inertia in a car racing game is that when the car turns left, the driver character in the car is pushed to the right.
- **Collision:** A collision is a phenomenon in which two or more objects collide with each other. An example of collision is a projectile fired from a weapon striking the target monster in a war game.

The physics concepts can be applied to the following physical systems:

- **Rigid body dynamics:** In physics simulation, rigid body dynamics is the study of how to represent and implement the animation of rigid bodies. A rigid body is a physical system that has finite size and that does not allow a change in its original shape when external forces are applied. For example, in a snooker game, the table, its pockets, and the balls are rigid bodies.
- **Soft body dynamics:** Soft body dynamics is the study of how to represent and implement the animation of soft bodies. A soft body is a physical system that has finite size and that changes its original shape when external forces are applied. You can apply soft body dynamics in your game if your game character makes use of cloth and that cloth

has to respond based on the movement of the game character. Soft body dynamics involves many more calculations than rigid body dynamics. Therefore, it requires a lot of computing power and is less frequently used in low-end systems.

- **Fluid dynamics:** Fluid dynamics is the analysis of animation of fluids. A fluid can be a liquid, such as water, or a gas. In games, you can apply fluid dynamics whenever you want to implement fluid effects similar to nature. For example, you might want to realistically imitate rising smoke or turbulent water flow. However, such simulation requires sound knowledge of mathematical skills and is expensive to implement.

PHYSICS ENGINE

In animation, we use a physics engine to implement a simulation. A ***physics engine*** is computer software that takes care of all mathematical calculations for implementing the physics concepts. It makes the task of a game developer easy by

- Manipulating the outcome when two objects in your game collide
- Applying gravity on the objects in your 3D game world
- Calculating the force applied on an object and manipulating the movement of that object based on the calculations

There are two types of physics engines, real-time and high precision. Real-time physics engines are not accurate, and so they are used in video games that require simplified calculations. High precision physics engines provide precise physics calculations and are usually used by scientists to create virtual simulators.

There are wide varieties of real-time physics engines available in the market for Microsoft XNA 4.0. Some of them are:

- Farseer Physics
- Jello Physics
- Box2Dx
- JigLibX
- BulletX
- BulletXNA

A physics engine generates a stream of animation points for a given animation information. The animation information can be data, such as the initial speed and angle of the animation. You can access a physics engine from the XNA Framework by creating a separate instance of the respective physics engine object. The following code sample accesses the `JigLibX Physics` engine.

```
//Include necessary library.

using JigLibX.Physics;
using JigLibX.Collision;
using JigLibX.Geometry;
using JigLibX.Math;
using JigLibX.Utils;

//Initialize Physics system

PhysicsSystem physicSystem;
//Initialize physics system
physicSystem = new PhysicsSystem();
//Initialize Collision System
 physicSystem.CollisionSystem = new CollisionSystemSAP();
//Initializing physics solver
 physicSystem.EnableFreezing = true;
 physicSystem.SolverType = PhysicsSystem.Solver.Normal;
```

```
physicSystem.CollisionSystem.UseSweepTests = true;
//Set time stamp for Collsion system
physicSystem.NumCollisionIterations = 1;
physicSystem.NumContactIterations = 1;

physicSystem.NumPenetrationRelaxtionTimesteps = 5;

/* Update physics system at 60fps for better output */
float timeStep = (float)state.DeltaTimeTicks / TimeSpan.
TicksPerSecond;
if (timeStep < 1.0f / 60.0f)
    physicSystem.Integrate(timeStep);
else physicSystem.Integrate(1.0f / 60.0f);
```

Like the GPU, which takes care of the processing of graphics in multimedia applications such as video games, today, a ***physics processing unit (PPU)*** is available in the market. A PPU is a dedicated microprocessor designed exclusively for handling physics calculations. The idea behind introducing PPU is to reduce the load on the CPU of performing time-consuming physics calculations. Some of the calculations that a PPU performs are rigid body dynamics, soft body dynamics, fluid dynamics, and collision detection.

A ***general-purpose computing on graphics processing unit*** (GPGPU) is another hardware processing unit that provides physics processing. A GPGPU is a GPU card that provides support for physics calculations. All graphics cards produced today have GPGPU.

Understanding Collision Detection and Response

Collision detection in games means determining whether objects or characters in a game world overlap each other. ***Collision response*** in games means determining the necessary action that should take place based on the nature of the collision involved.

It is impossible to develop a game that does not have collisions. When a projectile fired from a weapon strikes a villain character, a collision occurs. A collision also occurs when a player character hits a ball on a wall and the ball bounces back from the wall. From these examples, you can infer that collision detection is important in maintaining realism in games because detection prevents bodies from interpenetrating. In simple terms, collision detection helps you to decide whether there is any obstruction in front of your game character, such as a wall, or whether the character is going to walk into another object. Once a collision is determined, you can simulate the event that can happen because of the collision.

Collision detection involves a lot of mathematics, especially linear algebra and geometrical problems. You can determine a collision between two objects by calculating the time of impact and estimating the set of intersecting points. These calculations involve a lot of computing time, but because games do not need to imitate real physics, we can reduce the burden on the game player's computer system. However, although accuracy in physics is not necessary, the resulting simulation should be acceptable and should satisfy the game players.

To simplify the game developer's task of implementing collision detection systems in his or her games, Microsoft XNA 4.0 Framework provides classes and methods. The different classes provided by the XNA Framework represent three-dimensional areas—such as sphere and rectangular box or bounding box—for collision detection.

CERTIFICATION READY
How can you detect a collision between two game models using rectangular bounding box?
4.3

COLLISION DETECTION USING RECTANGULAR BOX

Collision detection using rectangular box involves enclosing 3D models within a rectangular bounding box and determining whether any two rectangular areas in your game world touch or overlap each other. The `BoundingBox` Structure available in the XNA Framework helps you create a bounding box that encloses your game model. A model in XNA is made up of

one or more meshes. Each face of the rectangular box is perpendicular to the X, Y, and Z axis. Let us now look at how you can determine the collision between two game models using the BoundingBox Structure.

DETECT COLLISION USING BOUNDING BOX

GET READY. Trace the position and the movement of each of your game models in the game world.

1. Create the bounding box for each model of your game scene using the BoundingBox class.

```
//Create bounding boxes
BoundingBox b1=new
  BoundingBox(minPointOfModel1,maxOintOfModel1);

BoundingBox b2=new
  BoundingBox(minPointOfModel2,maxOintOfModel2);
```

2. In the Game.Update method, transform the bounding box according to the model's world matrix.

```
//Get corner points of bounding box
  Vector3[] obb = new Vector3[8];
b1.GetCorners(obb);

/*Transform corner points according to model's world matrix*/
      Vector3.Transform(obb, ref model1WorldMatrix, obb);
//Create new bounding box from transformed points
  b1= BoundingBox.CreateFromPoints(obb);
```

3. Repeat Step 2 for the second bounding box as well.

```
//Get corner points of bounding box
  Vector3[] obb2=new Vector3[8];
b2.GetCorners(obb2);

/*Transform corner points according to model's world matrix*/
      Vector3.Transform(obb2, ref model1WorldMatrix, obb2);
//Create new bounding box from transformed points
B2= BoundingBox.CreateFromPoints(obb2);
```

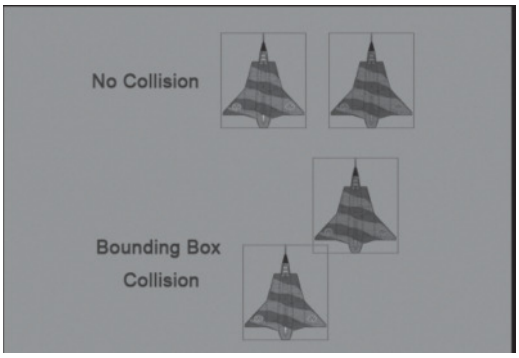
4. Check the collision using the function available in the BoundingBox class.

```
Public bool detectCollsion(BoundingBox
object1,BoundingBox object2)
{
/*Check collision detection .If both bounding box
intersects , then collision happens*/
         if(object1.Intersects(object2))
         {
                  return true ;
         }
         return false ;
}
```

One major disadvantage in using a bounding box is that when models do not fit exactly into your bounding box, it can lead to false positives for collision. In Figure 4-22, you can see that although the two game models do not touch each other, the two rectangular areas of the

Figure 4-22

An example of collision detection using the bounding box method



boxes intersect. To overcome this disadvantage in the bounding box collision detection method, you can use the per-pixel collision detection method.

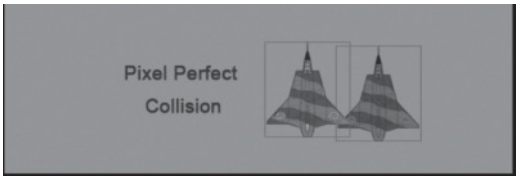
COLLISION DETECTION USING PER-PIXEL

CERTIFICATION READY
What is the concept behind the per-pixel collision detection method?
4.3

Images in computers are represented in pixels. The collision-detection method uses the overlap between pixels to detect collision. When the two rectangular areas of the bounding box intersect, every point within the intersecting bounds gets the color of the overlapping pixels. In addition, the overlapping pixels are no longer transparent, that is, if their alpha values are not equal to zero. These two findings confirm collision. Figure 4-23 shows collision detection using the per-pixel method.

Figure 4-23

An example of collision detection using the per-pixel method



TAKE NOTE*

Per-pixel collision is a kind of 2D collision detection technique. It is not used for 3D. It is a slow process, because you need to compare each pixel of texture. If the texture is large, then the number of pixels increases and collision detection time also increases. You need to use bounding box for collision detection for both 2D and 3D.

Once you have determined that two game models have collided into each other, you then decide the response that needs to be simulated following this collision.

COLLISION RESPONSE

CERTIFICATION READY
What is collision response?
4.3

When two game models in your game world collide, the resultant event depends on the nature of the collision. For example, consider a car racing game. When a speeding car faces an obstacle, such as a brick wall on the racetrack, the car immediately changes its direction to avoid collision. But if the two game models—the car and the brick wall—come into contact, the resultant event is that the car is slightly displaced from its position with a change in velocity. To cite another example, let us consider a bullet striking a villain's chest. In this case, the two game models involved in the collision are the bullet and the villain's chest. The villain lying on the ground indicating that he is dead is the resultant event of this collision.

Designing AI

Designing AI in video games includes methods to produce the delusion of intelligence in the behavior of ***nonplayer characters (NPC)***. Games are gaining commercial popularity with the inclusion of the AI feature.

Remember that physical simulation helps you construct approximate models of real-life physical systems in your games. You use physics concepts for your game objects to represent a range of physical activities, such as walking, turning, and changing facial expressions. Although physics-based animation helps you provide realistic motion to your NPCs, it does not provide a way to simulate human thinking. This is where AI comes into the picture.

In the past, game players mostly played against NPCs whose actions were hard-coded. These NPCs did not provide enough challenges to the game player because the movements of these game entities were predetermined and extremely limited. By repeatedly playing the game, players could easily predict the movement of the NPCs and, therefore, easily win the game. For example, in *Space Invaders*, the game player has to shoot the aliens before they reach the bottom to swoop down at the player. The movements of these aliens were hard-coded and not determined dynamically at run time. This was not quite challenging enough for the game players.

With the advent of different game genres, the need to simulate AI in NPCs has become important to engage the game players. To cite an example, a game that employs AI is *The Sims*. The characters in this game possess individual characteristics. They can act according to their will and have their own wants and needs.

The idea of applying AI in games is to simulate realistic intelligent behavior of NPCs similar to the behavior of a human. Creating viable AI is one of the biggest challenges for game developers because today, game players expect intelligent and difficult games. To meet their expectations, developers cannot ignore the AI component while developing games.

UNDERSTANDING AI

In general, the focus of AI in games is in the design of a believable challenging opponent for human players. The key role of AI in games is decision making. The AI in the game makes decisions using selective methodology and acts according to the selected option. Because you are aware of the game rules and the available game entities, you can provide scripted and predetermined data-driven algorithms and scripted responses to incorporate AI into your game. Therefore, the purpose of AI in games is not for intelligence, but for providing more realistic and challenging game entities for game players.

The main task behind creating a game AI involves understanding the requirement of the game outcome and then building the system to provide that outcome. The following content discusses some of the strategies in which you can incorporate AI in your games:

- **Most obvious action strategy**: You can build the AI in your game based on the most obvious action of the game player. For example, consider a first-person shooter (FPS) game in which the player needs to find a flower having special healing powers to complete a level. To complete this task, the player might need to overcome several challenges. For instance, he might need to escape from the attacks of an NPC. To escape the attack, the player character chooses to take cover behind some bushes. In this case, you can program the NPC to either wait for one minute and throw a bomb or randomly fire gunshots towards the bushes. Another option is to program the NPC to take a higher ground and shoot the player character hiding behind the bush. This strategy shows the simplest form of simulating AI in a game.
- **Characterization-based strategy**: Characterization-based strategy uses the nature of the behavior of the NPCs in a game. For example, if an NPC runs out of weapons to attack

the opponent player character, you can program the NPC to take cover or choose weapons from the store.

- **Condition-based strategy**: This strategy incorporates AI based on the current circumstance or condition, which the NPC is stuck with. For example, if an NPC is defeated by the player character and ends up in prison, you can program the NPC to escape from prison using the appropriate objects in the game environment. For instance, you can program the game character to use an explosive to break open the prison wall and escape through the breach in the wall. This way, you can improve the quality of AI in your game.
- **Path-finding strategy**: This strategy can be used to make an NPC find a path around an obstacle. For example, say in a role-playing game, the player character traps your NPC in a maze. You can program your NPC to move through the shortest possible route to get out of the maze. However, during this process, you should also consider the position of the player character and the possible obstacles from him. For instance, the player character could be firing at your NPC. In this case, you should also decide on how your NPC should react. Should she fire back or take cover? There are numerous pathfinding algorithms available that you can make use of in your game. A* is one such algorithm.
- **Rubberband strategy**: You can use this strategy in a racing or sports game. This strategy incorporates AI by preventing the player character from getting too far ahead of the NPC. For example, in a car racing game, you can program the NPC driver to speed the car in an effort to move ahead of the player character. This way, you can challenge the game player by maintaining the competitive feel.

Note that most games in some way involve chasing and escaping events. Keeping this in mind, you can design your AI algorithm considering the following points:

- While chasing the player character, use the most direct and shortest way for your game character. Keep the game character's line of sight straight.
- Choose an intercepting point between the player character and your game character based on his or her position, direction, and velocity.
- Use predefined complex patterns for movement of your game character. By using such patterns, you can provide the illusion of intelligent behavior to your game character.

The following steps show the basic flow of AI in an FPS game:

1. Decide which AI should move towards the player depending on the position.
2. Decide how the selected AI should behave depending on various factors, such as the action performed by the player, intelligence of the AI, and so on.
3. Use a path-finding algorithm to make the AI move towards the player.
4. Attack the player.

AI ENGINES

To design AI in your game, you can write your own algorithms or use AI engines that are specifically designed for XNA. In the same way that physics engines provide a complete solution for providing physics-based animation in your games, AI engines provide solutions that help you to simulate AI in your game. Examples of AI engines that are available for XNA include:

- SharpSteer
- SimpleAI
- XNA Pathfinding Library

CONSIDERATIONS FOR AI DESIGN

AI might appear unreal if overdone. No human player is perfect. Therefore, to support this realism, you should allow a certain amount of imperfection in your NPCs. Program your

NPCs in such a way that they make the game players believe that they are playing against real opponents rather than computer-simulated characters. For this, you have to allow your NPCs to make mistakes. The game players should be able to defeat your NPCs at some point in time. Perfect NPCs make the game unreal and boring by not giving the human player a chance to win.

SKILL SUMMARY

IN THIS LESSON YOU LEARNED:

- Gameflow is a sequence of challenging tasks and corresponding rewards that make players enjoy the game and give them a sense of accomplishment.
- Game states are characterized by a combination of visual, audio, and animation cues that make the game go forward.
- An optimized game state ensures that all elements in the game state are utilized efficiently.
- Scene hierarchy helps you design a game that requires less memory to run smoothly.
- Fps-induced delays or hindrances to members disrupt the gameflow.
- A graphics pipeline refers to the current method of drawing images on a screen through the conversion of vector graphics format or geometry into raster or pixel images.
- The main loop is a part of code that gets repeated throughout the duration of the game. In XNA, `Game.Draw` and `Game.Update` form the main loop.
- The `Update` method is used whenever the game logic needs to be processed.
- The `Draw` method is used whenever it is time to draw a frame.
- In the fixed step type, the game code calls the `Update` method at fixed intervals.
- In the variable step game loop, the game code calls the `Update` and `Render` or `Draw` methods continuously.
- The various processes involved in designing an object or a character when developing a game include creating complex objects from simple objects, distorting an object when required, making the object move, creating plane structures, and applying a suitable interpolation method to animate the character appropriately.
- The process of animation requires careful attention and skill to bring out the liveliness of a specific character.
- The various functions available for animating a character include movement techniques such as walking, rotating, and shifting from one place to another on the screen space; monitoring the interpolation to a specified duration for a particular frame; applying sprite animation to create the visual effects in a given scenario; and transforming the 3D graph of an object into the gaming world using matrices.
- It is easy to create a feel of the character in the XNA Framework using lighting because XNA has a built-in formula to calculate the amount of light that needs to be reflected off of a surface.
- The XNA Framework allows for per-pixel lighting, which ensures a smooth lighting effect on the surface.
- A filter is a method or technique that enables you to change the appearance of textures on images.
- The `SamplerState` class allows you to set the parameters for magnification and shrinkage of textures using the `Filter` property.
- A shader is the surface property of an object.
- The vertex shader takes the 3D position of the vertex and transforms it into 2D screen coordinates.
- The pixel shader draws each pixel of the screen using the coordinates it receives from the vertex shader.

- Projections effect scaling and view changes to the character or object's environment.
- The XNA Framework allows you to create custom-made objects through the use of the `DrawUserIndexedPrimitives` class object.
- You can use physics simulation in your game to provide realistic animation effects.
- A real-time physics engine helps game developers implement physics in their game animation.
- Some of the different physics engines used for XNA include Farseer Physics, Jello Physics, Box2DX, JigLibX, BulletX, and BulletXNA.
- A PPU is a dedicated microprocessor designed exclusively for handling physics calculations.
- A GPGPU is a GPU card that provides support for physics calculations.
- Collision detection in games involves determining whether two game models in your game world have contacted each other.
- Collision response involves determining the event that needs to be simulated based on a collision.
- Bounding box collision detection and per-pixel collision detection are methods for detecting collisions in your game.
- AI in games focuses on the design of believable challenging opponents for human players.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. Providing different styles of playing the game at each level maintains the _____ of the game.
2. The _____ type of game loop is best suited for time-based movement and animation operations.
3. You must use _____ to control the gameflow.
4. The _____ method needs to be overridden for game-specific rendering code.
5. You can use the _____ function of the `MathHelper` class to calculate the distance between objects.
6. You can use the _____ matrix to enlarge objects when creating a game model.
7. The smallest unit of texture that a GPU can process is called a(n) _____.
8. A(n) _____ is used to shrink a texture without causing jagged edges on the final image.
9. _____ is a microprocessor that is designed exclusively for performing physics calculations.
10. _____ collision detection method checks for the transparency of pixels.

Multiple Choice

Circle the letter or letters that correspond to the best answer.

1. Which of the following are features of gameflow? (Choose all that apply.)
   a. Challenge
   b. Performance

c. Scripted events
d. Trial and error
e. Player commands
f. Player vocabulary

2. Which of the following are stages in the graphics pipeline? (Choose all that apply.)
a. Visibility
b. Resizing
c. Resolution
d. Transform and lighting

3. Which of the following is a game state in which the elements are utilized properly?
a. Start
b. Optimized
c. Forward
d. End

4. Which of the following would you use in a game code that requires the game loop to be based on `TargetElaspedTime`?
a. Update
b. Fixed step
c. Variable step
d. Initialize

5. Which of the following are methods available for the main loop with the XNA Framework? (Choose all that apply.)
a. Update
b. Draw
c. Interpolate
d. Initialize

6. Which of these methods are associated with the `Plane` class? (Choose all that apply.)
a. Equals
b. Translates
c. Transform
d. Intersects

7. Which of the following will you use in the `SamplerState` class if the texture is too big or too small for your image?
a. Mipmaps
b. Filters
c. Projection matrix
d. Shaders

8. Which of the following are filtering methods available with the XNA Framework? (Choose all that apply.)
a. Point filtering
b. Linear filtering
c. Mipmap filtering
d. Anisotropic filtering

9. Which of the following is *not* a physics engine?
a. Farseer
b. Box2D
c. Microsoft XNA 4.0 Game Studio
d. BulletXNA

10. You decide to use physics simulation in your game animation. Which of the following will help you to implement physics simulation?
   a. Real-Time Physics Engine
   b. High Precision Physics Engine
   c. Microsoft XNA 4.0 Game Studio
   d. All of the above

■ Competency Assessment

Scenario 4-1: Defining Gameflow

You are developing an FPS game. Define the various possible features of the gameflow for your game.

Scenario 4-2: Tracking Game Levels

You have different levels in your fantasy game. Explain the possible states of your game in a particular level.

■ Proficiency Assessment

Scenario 4-3: Creating Shadow

In your adventure game, you are required to provide an illusion of shadow of the game objects. Explain the mechanism to achieve this with a sample code.

Scenario 4-4: Detecting Collision

You use physics-based simulations in your game. Explain the mechanism to detect collisions between your game objects with appropriate code using XNA 4.0.

5 LESSON

Developing the Game User Interface (UI)

EXAM OBJECTIVE MATRIX

SKILLS/CONCEPTS	MTA EXAM OBJECTIVE	MTA EXAM OBJECTIVE NUMBER
Managing the UI Assets	Design the user interface. Capture user data. Work with XNA. Plan for game state.	1.4 1.6 1.7 3.2
Programming the UI Game States	Design the user interface. Work with XNA.	1.4 1.7
Programming the UI Access Mechanisms	Design the user interface. Work with XNA.	1.4 1.7

KEY TERMS

`Microsoft.Xna.Framework.Input` **namespace**
save-load UI
UI access mechanisms
UI asset

Steve Watson works at Constoso Gaming Inc., and leads the development team. His company is developing a first-person shooter (FPS) game for console and computer systems. The members of the team have designed the game's visual world. They have also designed various components for their game using XNA 4.0. They are now in the process of developing code to load or program their game's user interface (UI) assets in a specific sequence as per the gameflow.

In this development phase, the team members correlate the required audio and video assets and player's input with the UI assets. The team also decides to define the behavior of the UI assets using game states. They also programmatically create the UI access mechanisms for their game.

■ Managing the UI Assets

THE BOTTOM LINE

The user interface (UI) designed for a game is made functional in the game through programming. To program the functionality of the UI in your game, you need to load the UI into your game through the XNA Framework content pipeline.

CERTIFICATION READY
What are the different activities involved in managing your game's UI assets?
1.4

As you can recall, *UI* is a collective term used to refer to the onscreen elements through which a player interacts with the game. The UI helps the player access information about the game world and the status of his or her character. As you learned in Lesson 3, "Creating the Game Output Design," the UI concept is carefully thought out and the design of the UI is envisioned during the game's design phase. The ***UI assets*** designed during that phase—such as menu, sprites, and GUI controls—need to be created or loaded in the XNA Framework before you can add code to make them functional in your game. In other words, managing the UI assets involves adding the UI assets designed for the game into the game environment so that the user can access them in the sequence envisioned during the design phase.

Managing the UI assets might involve the following activities:

- Loading the existing UI assets, such as an image or sprites, on the game screen
- Configuring audio, video, and player inputs for the UI assets
- Creating menus
- Creating the UI for save-load

Let's look at these activities in detail.

Loading the UI Assets

Loading the UI assets means adding the code to create the UI assets and making them available for the user. It does not include making the UI assets functional at this stage. The functionality is added at the next stage.

CERTIFICATION READY
How will you access the existing UI asset in your game?
3.2

UI plays an important role in any game. For example, while playing the game, the player might need to know about his health status, score, amount of ammunition left, and so on. You can provide this information to the player through the UI. For instance, you can use a health bar texture to indicate health, text to display the score, and a menu to help the player select the required weapon or inventory of his choice. Alternately, you can load other game models to the game environment. These art and data assets can be in different formats such as .jpg, .fbx (Autodesk FBX file format), or .spritefont.

To access these game assets at run time for immediate use, XNA Framework 4.0 provides you with the content pipeline. You can use the art or data files created by a game artist directly in your XNA Game Studio projects by loading them into the content pipeline. The content pipeline allows you to include the necessary game assets in the form of managed code object in the game's executable. To make your game assets available to the content pipeline, you need to load the required asset to the game content project. The following section describes the steps required to load a UI asset to a game content project.

LOAD UI ASSET

GET READY. Create your game project and name it "SampleGame." Add the graphics elements or the UI assets to your game content by performing the following steps:

1. In Solution Explorer, right-click the **Content** project and click **Add > Existing Item** (see Figure 5-1). Then click **Existing Item** and browse to your saved graphics. Add the necessary UI elements, such as sprites, images, and so on. This example selects the Background.jpg image file (see Figure 5-2). Figure 5-3 shows the Content node listing the Background.jpg file. Note that you can store the relative graphics content in distinct folders for the ease of management of assets.

   For example, you can create a folder called *Backgrounds* and add all background images to it. For a particular scene in the game, you might use a distinct folder called *scene1*

Figure 5-1

The Solution Explorer showing the Add option

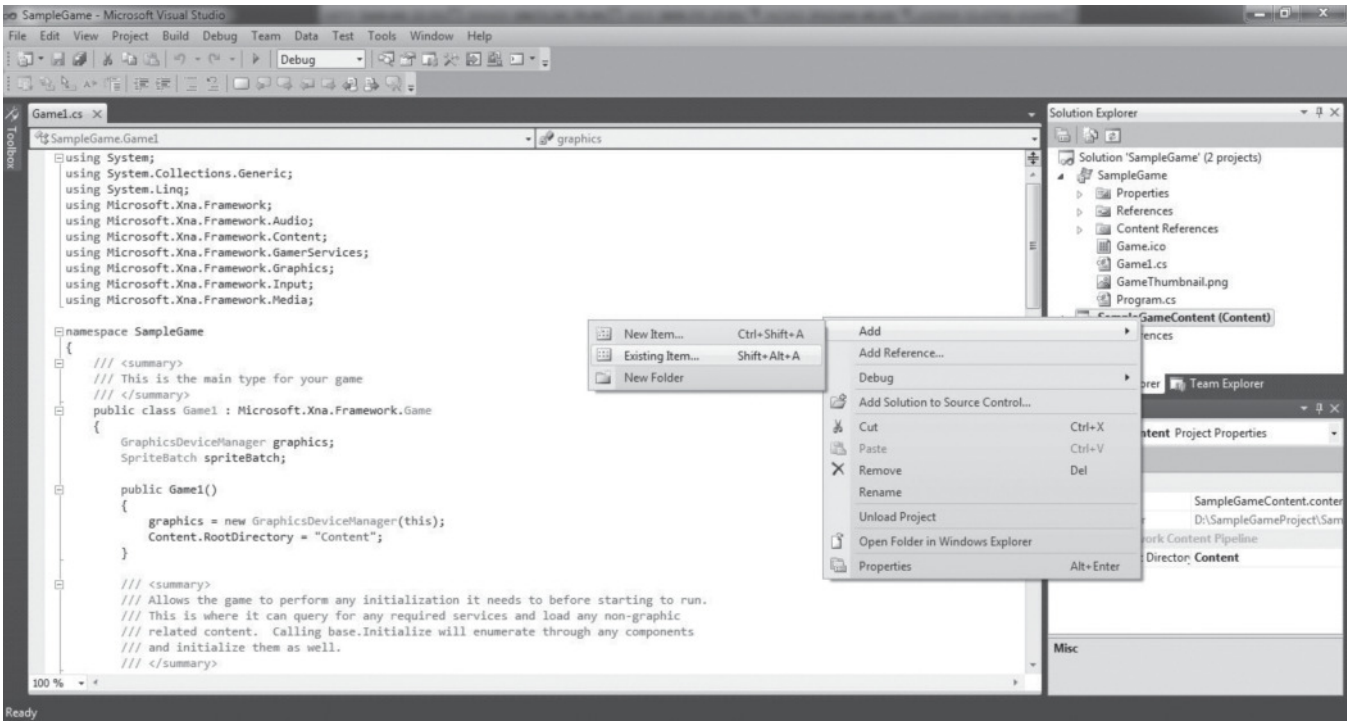


Figure 5-2

The Add Existing Item dialog box listing the image file

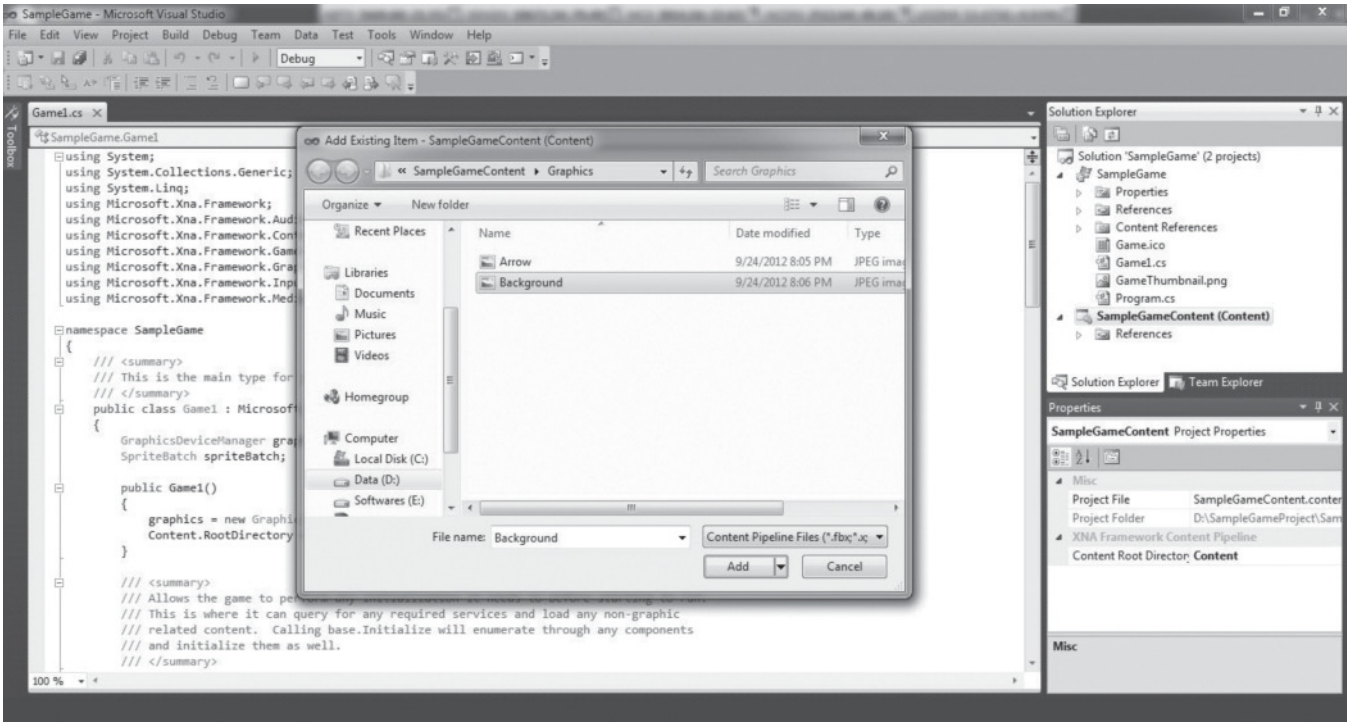
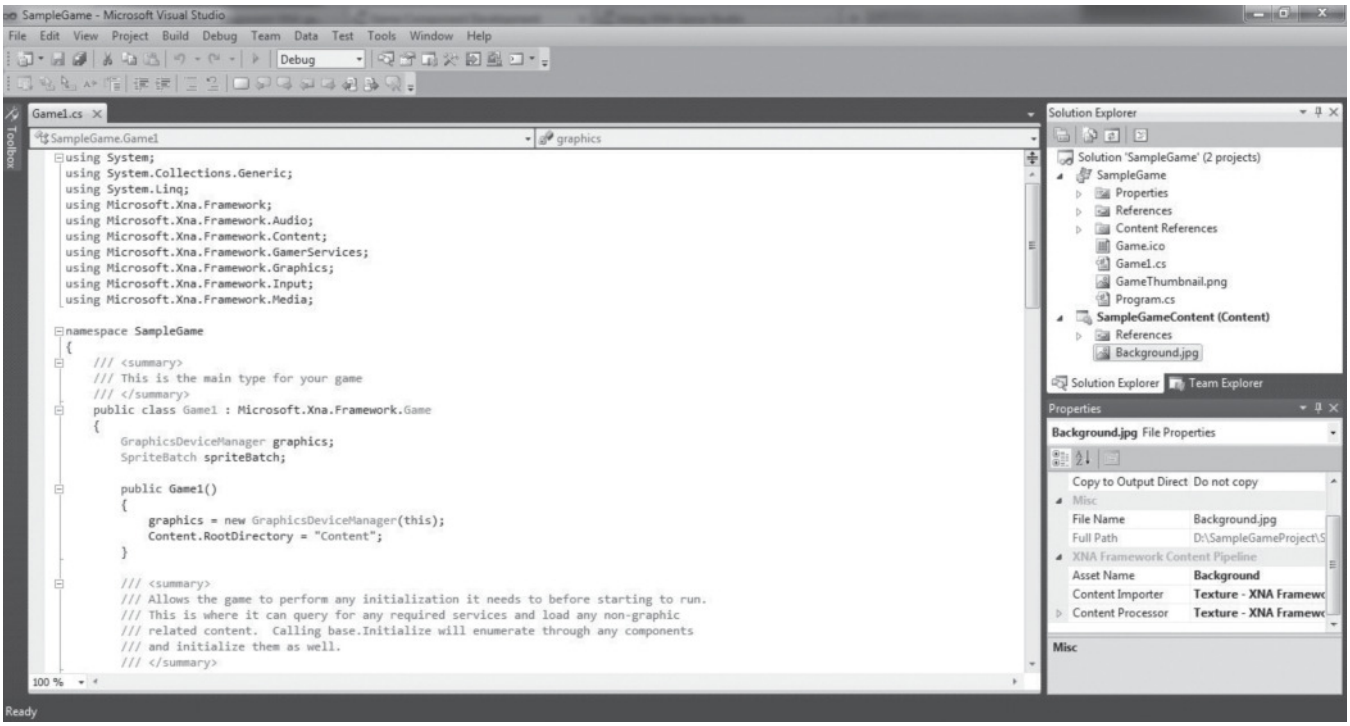


Figure 5-3

The Content node listing the image file



and add all scene1 data inside it. Maintaining the game content in distinct folders helps you manage the assets in an efficient manner.

2. Once you add the required assets, you can load them onto the game code by overriding the `LoadContent` method, as shown in the following code. The `LoadContent` method loads the required assets into your game. The following code loads an image file for setting the background of a game's title screen. Note that the code uses "@" before the string parameter of the `Content.Load` method. The use of "@" does not allow "\" escapes and is thus much easier to define the path to a file.

```
protected override void LoadContent()
{
/* the Content.Load method loads a game asset that has been
processed by the content pipeline */
Texture2D backgroundScreen =
Content.Load<Texture2D>(@"Background");
Base.LoadContent();
}
```

TAKE NOTE*

When you add a graphics file to your game content project, XNA Framework 4.0 automatically stores it in the content pipeline. This helps you in loading the required graphics in your game through the code.

Now that you have seen how to load an existing graphics asset into your game, let us now look at how to configure sound, video, and player inputs with the required UI asset in your game.

CERTIFICATION READY

How would you load a background image to your game in XNA 4.0?

1.7

Configuring Options

Correlating the audio, video, and the player input with the UI assets is important in any game.

CERTIFICATION READY
What are the ways to associate audio, video, and keyboard inputs with the UI assets of your game?
3.2

For example, if the player clicks a button, the player feels good if she hears a click sound. Similarly, when the player clicks on a game avatar, it might be a good idea to display a video in which the avatar speaks to the player listing the different avatar options. Additionally, you might also need to capture the keys that a player presses for activating the specified UI asset. Let us look at these configuration options.

CONFIGURING THE AUDIO FILE

Let us understand how to configure sound effects in your game. For example, while pressing any UI button, the player might hear a click sound, which helps the player to understand that the button has been clicked. Other examples of using sound in a UI are:

- When a player is choosing an avatar, the avatar might speak a dialog.
- When the player has a menu displayed, she might be able to hear an ambient background track playing until she enters the gameplay.

TAKE NOTE*

Audio in XNA Framework 4.0 is wave-based. You can use the `SoundEffect` class to use wave files in your game.

XNA Framework 4.0 provides several audio classes in the `Microsoft.Xna.Framework.Audio` namespace, which you can use to play back audio. The `SoundEffect`, `SoundEffectInstance`, and `DynamicSoundEffectInstance` classes provide everything that you need to play and stream audio during gameplay.

The easiest way to incorporate background sound effects for your UI assets is by using the `SoundEffect` class. You can add audio files just like you add any other game asset to the project. The following steps discuss the procedure to add audio files to your game.

ADD SOUND EFFECTS

GET READY. In order to add sound effects to your game project, do the following:

1. In Solution Explorer, right-click the **Content** project and then click **Add**. Click **Existing Item** and browse to the location where you have saved the wave files and select the item.
2. Load the wave files in your game using the `LoadContent` method.

The following code performs the previous steps:

```
//Set the sound effects to use
SoundEffect soundOnMenuDisplay;
//Modify the Loadcontent method

protected override void LoadContent()
{
        /* Create a new SpriteBatch, which can be used to draw
textures.*/
        spriteBatch = new SpriteBatch(GraphicsDevice);
        soundOnMenuDisplay =
Content.Load<SoundEffect>(@"your sound file name");
/;
}
/*Now in Game Update loop you can play the audio for the
respective UI asset*/
protected override void Update(GameTime gameTime)
        {
```

```
/*Write code to display the menu and play the background sound
using the soundOnMenuDisplay.Play() */

        base.Update(gameTime);
        }
```

CONFIGURING THE VIDEO FILE

Like audio, you can also add videos to the UI. For example, you might want to include a video of an avatar speaking a dialog when presenting the avatar options for the player to choose from. You might even want to include a video in the background of the UI that simply runs to give the player a glimpse of what exactly will happen in the game.

You can add video components just like any other UI assets using the `LoadContent` method. XNA Framework 4.0 allows video components in Windows Media Video format. You simply have to add the video files to a distinct folder for ease of accessibility and then load it in the game code using the `LoadContent` method.

Once you load the required video file to the content pipeline, you can represent the video file using the `Video` class provided in the `Microsoft.Xna.Framework.Media` namespace. This namespace also provides a `VideoPlayer` class to play, pause, and resume the video.

MORE INFORMATION

For more information on the capabilities of the XNA Game Studio for providing the video playback, refer to the "Media Overview" section in the MSDN Library.

The following code sample displays a video. The code creates an object of the `Video` class to represent the specified video file. It then creates a `VideoPlayer` object to provide the functionality of playing, pausing, and resuming the video. The code uses the XNA Sprite to draw the current frame of video on the screen.

MORE INFORMATION

For more information on `Video` and `VideoPlayer` classes, refer to the `Microsoft.Xna.Framework.Media` section in the MSDN Library.

```
        /*Define an object for video player and video .*/

        Microsoft.Xna.Framework.Media.VideoPlayer videoPlayer;
        Microsoft.Xna.Framework.Media.Video videoObject;

        /*.Initialize video player in Game.Initialize()*/

        videoPlayer = new
Microsoft.Xna.Framework.Media.VideoPlayer();

        /*Load the media file you want to play in video player in
Game.LoadContent()*/

        videoObject=
content.Load<Microsoft.Xna.Framework.Media.Video>(@"your video
file path");

        /* You can play and stop video using following
statements.*/

        videoPlayer.Play(videoObject);
        videoPlayer.Stop();
```

```
/*To draw the video on the screen*/
public void Draw(DrawState state)
      {
            Vector2 pos = new Vector2(10.0f,10.0f);
            Texture2D texture = videoPlayer.GetTexture();
//Now draw it using xna sprite
spriteBatch.Begin();
spriteBatch.Draw(texture, pos, Color.White);
spriteBatch.End();

}
```

CONFIGURING PLAYER INPUTS

CERTIFICATION READY
What is the mechanism behind acquiring the input from the player in your FPS game?
1.6

You can retrieve user inputs from the respective input device such as a mouse or a keyboard to map the captured data to the desired UI asset. For example, in an FPS game, you can map the captured keystrokes with the action of firing the bullets from a gun and play an audio file to provide the sound effect of the gunshots. Alternately, to detect which menu item the user has selected using the keyboard or a mouse, you need to retrieve the keystrokes or mouse clicks generated from the keyboard or a mouse. The mechanism behind acquiring the input from the player and then using it to trigger certain actions is called *handling the input states*. XNA 4.0 includes all the functionalities required to capture the current state of the input devices, such as the keyboard, mouse, and joystick. You simply need to access the specific classes and structures provided in the ***`Microsoft.Xna.Framework.Input` namespace***. The `Keyboard`, `Mouse`, and `Gamepad` classes help to retrieve the keystrokes from the keyboard, button clicks from the mouse, and button presses on the Xbox360 controller, respectively. The structures in the `Input` namespace help you to determine the current state of the input. For example, the `KeyboardState` structure stores the current state of the keyboard and helps you to determine the keystrokes generated from the keyboard.

Table 5-1 presents some of the input structures available in the `Microsoft.Xna.Framework.Input` namespace.

Table 5-1
Structures in the Input Namespace

Structure	Description
`GamePadButtons`	Identifies whether buttons on the Xbox controller are pressed or released
`GamePadCapabilities`	Identifies the capabilities and type of Xbox controller
`GamePadState`	Describes the current state of Xbox controller
`GamePadThumbSticks`	Represents the position of left and right sticks
`MouseState`	Represents the current state of the mouse
`Keyboardstate`	Represents the state of keystrokes recorded by a keyboard

Handling input states involves three basic tasks:

- Gathering the raw input from the input device.
- Mapping the input and connecting the input with the appropriate input handler.
- Defining the input handler code that will direct the game as to what action to do and when to do it.

The `Microsoft.Xna.Framework.Input` namespace also provides enumerations for keys and game pad buttons. However, it does not provide any enumeration for the mouse. You need to define the enumeration for mouse buttons in your game code.

MORE INFORMATION

For more information on `Input` namespace, refer to the "`Microsoft.Xna.Framework.Input`" section in the MSDN Library.

The following sections discuss the procedure required to capture the state of the different input devices.

DETECTING THE STATE OF KEYS

Players can use the keyboard to issue commands to the units in a game much more quickly than by clicking on the icons on the screen. For example, in an FPS game, players can press the "A" key continuously for shooting bullets from a gun. Perform the following steps required for detecting a key press using the `Keyboard` class.

DETECT A KEY PRESS

GET READY. Create your project and name it `XNAKeyboardHandler`.

1. Declare an instance of type `KeyboardState` class, say LastKeyboardState, and allow it to hold the value of the last state of the keyboard. Similarly declare another instance of `KeyboardState`, such as `CurrentKeyboardState`, to hold the value of the current state of the keyboard.
2. Assign a value to `LastKeyboardState` in the game constructor.
3. Call the `GetState` method to hold the current keyboard state.
4. Compare the values of the two keyboard states.
5. Update the `LastKeyboardState` to hold the current keyboard state. This step ensures that the value of the current state is always compared with the value of the previous frame state.

   The following is the sample code for the previous steps.

```
namespace XNAKeyboardHandler
{
    class KeyboardHandler
    {
        /*define instances of KeyboardState structure the
current and previous keyboard states */

        public KeyboardState currentKeyboardState;
        public KeyboardState lastKeyboardState;

        public KeyboardState CurrentState
        {

            get
```

```
            {

                return currentKeyboardState;
            }

        }
        public KeyboardState LastState
        {

            get
            {

                return lastKeyboardState;
            }
         }

         public KeyboardHandler()
         {
          /* Save the current keyboard state */

             currentKeyboardState = new KeyboardState();
         }
         public void Update(GameTime gameTime)
         {
          /* get the new keyboard state and save the old
keyboard state */

             lastKeyboardState = currentKeyboardState;
             currentKeyboardState = Keyboard.GetState();
         }
         public void clearPrevState()

         {
             lastKeyboardState = currentKeyboardState;
         }

          /* Check for any key release */
         public  bool KeyReleased(Keys key)
         {

             return currentKeyboardState.IsKeyUp(key) &&
             lastKeyboardState.IsKeyDown(key);
         }

          /* Check for any key press */
         public  bool KeyPressed(Keys key)
         {

             return currentKeyboardState.IsKeyDown(key) &&
             lastKeyboardState.IsKeyUp(key);
         }

 /* check for key down */
         public  bool KeyDown(Keys key)
         {

             return currentKeyboardState.IsKeyDown(key);
         }
     }
}
```

DETECTING THE STATE OF MOUSE

Perform the following steps to detect the current state of a mouse input device.

GET THE MOUSE POSITION

GET READY. Create your project and name it `XNAMouseHandler`.

1. Call `Mouse.GetState` to get the current state of the mouse.
2. Use `MouseState.X` and `MouseState.Y` to get the position of the mouse in pixels.

The following sample illustrates the previous steps. The following code creates a helper class that gets the state of the mouse buttons and the position of the mouse on the game screen.

```
namespace XNAMouseHandler
{

    public class MouseCursor
    {

         /* texture for mouse cursor */
        Texture2D mouseCursor;
         /* sprite batch to draw the mouse texture */

        SpriteBatch spriteBatch;

        MouseState previousMouseState,currentMouseState ;

        public MouseCursor(Texture2D texture, SpriteBatch
spriteBatchRef)
       {
            mouseCursor = texture;
            spriteBatch = spriteBatchRef;
          /* save current mouse state */
            currentMouseState = Mouse.GetState();
       }

       public void update(GameTime gameTime)
       {
         /* Get the new mouse state and compare with the cur-
rent mouse state. If there is a difference, store the new mouse
state as current mouse state */
            if (Mouse.GetState() != currentMouseState)
            {
                previousMouseState = currentMouseState;
                currentMouseState = Mouse.GetState();
            }

       }
/* check for left button press */
       public bool isLeftButtonPressed()
       {

           currentMouseState = Mouse.GetState();

           if (currentMouseState.LeftButton == ButtonState.
Pressed)
            {
                return true;
            }
```

```
            return false;
        }
/* Check for right button press */
        public bool isRightButtonPressed()
        {
            currentMouseState = Mouse.GetState();
            if (currentMouseState.RightButton == ButtonState.
Pressed)
            {
                 return true;
            }
            return false;
        }
/* check for middle button press */
        public bool isMiddleButtonPressed()
        {
            currentMouseState = Mouse.GetState();
            if (currentMouseState.MiddleButton == ButtonState.
Pressed)
            {
                 return true;
            }
            return false;
        }
/* Check if middle button is released */
        public bool isMiddleButtonReleased()
        {
            currentMouseState = Mouse.GetState();
            if (currentMouseState.MiddleButton == ButtonState.
Released)
            {
                return true;
            }
            return false;
        }
/* check if left button is released */
        public bool isLeftButtonReleased()
        {
            currentMouseState = Mouse.GetState();
            if (currentMouseState.LeftButton ==
ButtonState.Released)
            {
                return true;
            }
            return false;
         }
/* check if right button is released */
         public bool isRightButtonReleased()
         {
```

```
            currentMouseState = Mouse.GetState();
            if (currentMouseState.RightButton ==
ButtonState.Released)
            {
                return true;
            }
            return false;
        }
/* checking a mouse button click is bit tricky . You can do it
by checking currentMouseState and previousMouseState. If
currentMouseState is ButtonState.Released and previousMouseState
is ButtonState.Pressed, then a mouse button is clicked */
/* Check if left button is clicked  */
        public bool isLeftButtonClicked()
        {
            currentMouseState = Mouse.GetState();
            if (currentMouseState.LeftButton ==
ButtonState.Released &&
                 previousMouseState.LeftButton ==
ButtonState.Pressed)
            {
                 return true;
            }
            return false;
        }
/* check if right button is clicked */
        public bool isRightButtonClicked()
        {
            currentMouseState = Mouse.GetState();
            if (currentMouseState.RightButton ==
ButtonState.Released &&
                 previousMouseState.RightButton ==
ButtonState.Pressed)
            {
                 return true;
            }
            return false;
        }
/* check if middle button is clicked */
        public bool isMiddleButtonClicked()
        {
             currentMouseState = Mouse.GetState();
             if (currentMouseState.MiddleButton ==
ButtonState.Released &&
                previousMouseState.MiddleButton ==
ButtonState.Pressed)
            {
                return true;
            }
```

```
            return false;
        }

        public void Draw(GameTime gameTime)
        {

            /* Draw the mouse cursor at the mouse position */
            MouseState currentMouseState = Mouse.GetState();
            Vector2 mousePos = new Vector2(currentMouseState.X,
currentMouseState.Y);

            spriteBatch.Begin();
            spriteBatch.Draw(mouseCursor, mousePos, Color.White);
            spriteBatch.End();
        }
    }

}
```

The following code declares an instance of the helper class in the game class and displays a message about the state of the mouse buttons whether clicked or pressed:

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

/* declare an object of MouseCursor class to track all mouse
operation */

        MouseCursor mouseCursor;


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

        protected override void LoadContent()
        {

            // Create a new SpriteBatch, which can be used to
draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);
spriteFont = Content.Load<SpriteFont>(@“Arial”);

            //initialize Mouse cursor
            Texture2D mouseCursorTexture = Content.Load
<Texture2D>(@“Arrow”);
            mouseCursor = new MouseCursor(mouseCursorTexture,
spriteBatch);

        }
```

```
        protected override void Update(GameTime gameTime)
        {

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

                this.Exit();
            /* update the mouse cursor */
            mouseCursor.update(gameTime);

            base.Update(gameTime);
        }
        protected override void Draw(GameTime gameTime)
        {

            GraphicsDevice.Clear(Color.CornflowerBlue);

            /* draw the mouse cursor */
            mouseCursor.Draw(gameTime);

 /*Display text according to current state of mouse */

            spriteBatch.Begin();
            if (mouseCursor.isLeftButtonClicked())
            {

                spriteBatch.DrawString(spriteFont, "Left Button
is clicked ", new Vector2(20.0f, 300.0f), Color.Yellow);
            }
            if (mouseCursor.isLeftButtonPressed())
            {

                spriteBatch.DrawString(spriteFont, "Left Button
is pressed ", new Vector2(20.0f, 300.0f), Color.Yellow);
            }
            spriteBatch.End();
            base.Draw(gameTime);
        }
    }

}
```

DETECTING THE STATE OF XBOX 360 CONTROLLER

The following steps illustrate how to determine the state of an Xbox input controller device using a sample code.

DETERMINE THE XBOX BUTTON PRESSES

GET READY. Create your project and name it `XboxController`.

1. Use `GetState` to determine the current state of the Xbox.
2. Verify whether the Xbox is connected using the `IsConnected` property.
3. Get the values of the buttons you want to check if pressed currently. For any button, if the value is `Pressed`, it means the button is currently pressed by the player.

```
namespace XboxController
{
    class XNAGamePad
    {
/* define instances of GamePadState structure. These instances
will hold the information of XBOX controller */
        GamePadState previousGamepadState, currentGamepadState;
        public XNAGamePad()
        {
/ To get the state of GamePad you need PlayerIndex that speci-
fies which GamePad's state is queried. XBOX supports multiple
GamePads at same time */
            currentGamepadState =
GamePad.GetState(PlayerIndex.One);
        }
        public void update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One)
!=currentGamepadState)
            {
                previousGamepadState = currentGamepadState;
                currentGamepadState = GamePad.GetState
(PlayerIndex.One);
            }
        }
/* Check for specific button press */
        public bool isButtonPressed(Buttons btn )
        {
            currentGamepadState = GamePad.GetState
(PlayerIndex.One);
            if (currentGamepadState.IsConnected &&
currentGamepadState.IsButtonDown(btn))
            {
                return true;
            }
            return false;
        }
        /* Check for specific button release */
        public bool isButtonReleased(Buttons btn)
        {
            currentGamepadState = GamePad.GetState
(PlayerIndex.One);
            if (currentGamepadState.IsConnected &&
currentGamepadState.IsButtonUp(btn))
            {
                return true;
            }
            return false;
```

```
        }
       /* check for specific button clicks */
        public bool isButtonClicked(Buttons btn)
        {
            currentGamepadState =
GamePad.GetState (PlayerIndex.One);
            if (currentGamepadState.IsConnected &&
currentGamepadState.IsButtonUp(btn) &&
previousGamepadState.IsButtonDown(btn))
            {
                return true;
            }
            return false;
        }
    }
}
```

The following code sample captures the key press "A" and plays an audio file:

```
/* declare an object of SoundEffect and an object of
KeyboardHandler in your Game class in the XNAKeyboardHandler
project */

SoundEffect soundOnKeyPressA;
KeyboardHandler keyHandler;

/* modify the LoadContent method as shown below */

protected override void LoadContent()
{

    /* create a new SpriteBatch, which can be used to
draw textures */
    spriteBatch = new SpriteBatch(GraphicsDevice);

    soundOnKeyPressA =
Content.Load<SoundEffect>(@"your sound file name");
}

/* Game.Update loop */
protected override void Update(GameTime gameTime)
{
    /* Call the Update method of the KeyboardHandler to get the
current state of the keys */
            keyHandler.Update(gameTime);

    /* Check for key press 'A' */

            bool isDown = keyHandler.KeyDown(Keys.A);

            if (isDown)
            {
                soundOnKeyPressA.Play();
            }

            base.Update(gameTime);
    }
```

MORE INFORMATION

For more information on handling the different input devices in your game, refer to the "Responding to User Input" section in the MSDN Library.

CERTIFICATION READY
What are the different classes and structures available in XNA Framework 4.0 to handle player inputs from different input devices?
1.7

Creating Menus

Menus are an integral part of the UI of every multimedia application, including games.

As you already learned in Lesson 3, you can use menus to provide players with a list of options. You can use menus as part of the game story or the game space. For example, you can use menus in the game story to provide options for the player character to select a particular weapon or an inventory. You can also use menus as a nondiegetic component on a welcome or opening screen, where players can select the activity they want to perform.

CERTIFICATION READY
How will you create menus for your game in XNA 4.0?
1.7

PROGRAM MENUS

You can create menus for your game in different ways. One way is to create the menu as a drawable game component and then add the component to your game's content solution to access it in the code. A drawable game component provides a modular approach to adding graphics content to your game. You can register the drawable game component with your game class by passing the component to the `Game.Components.Add` method. Once you register the component, the component's initialize, draw, and update methods are called automatically from the `Game.Initialize`, `Game.Draw` and `Game.Update` methods.

MORE INFORMATION

For more information on the drawable game component class, refer to the "`Microsoft.Xna.Framework`" section in the MSDN Library.

Create your project and name it *CustomMenu*. To create a drawable component, perform the following steps:

1. In Solution Explorer, select **Add** and then select **New Item**.
2. In the Add New Item dialog box, select the `GameComponent` and name it `MenuComponent` (see Figure 5-4).
3. Once the class is generated, change the base class from `Microsoft.Xna.Framework.GameComponent` to `Microsoft.Xna.Framework.DrawableGameComponent`.

This automatically generates the default constructor for the `MenuComponent` class, an override for the initialize method, and an override for the update and the draw methods. Now, you need to add the appropriate code to create the required menu items and make them appear on the screen. The following steps create a simple menu interface that lists different options for the player, such as options to start the game, view high scores, and end the game.

CODE A MENU

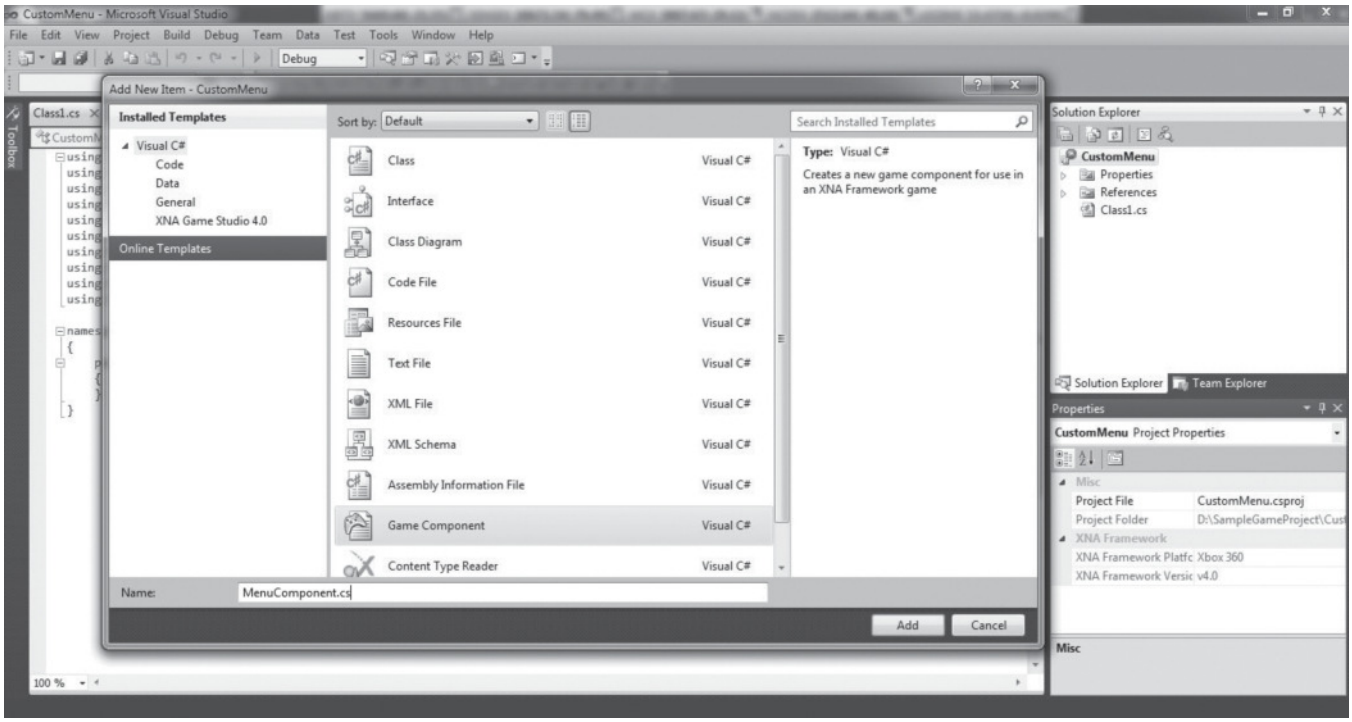
GET READY. Create the required menu items and draw them on the screen.

1. Create a `MenuItem` class to hold each menu item, as shown in the following code:

```
namespace CustomMenu
{
    class MenuItem
    {
        public string menuName;
```

Figure 5-4

The Add New Item dialog box listing GameComponent




```
        public Vector2 position;
       /* defines the normal color of the menu item */
        public Color normal = Color.White;
       /* defines the color of the menu item when it is
highlighted */
        public Color highlight = Color.Yellow;
       /* constructor that initializes the name and the posi-
tion of the menuitem */
        public MenuItem(string name,Vector2 pos)
        {
            menuName = name;
            position = pos;
        }
    }
}
```

2. Write the following code in the MenuComponent class:

```
namespace CustomMenu
{
public class MenuComponent : Microsoft.Xna.Framework.
DrawableGameComponent
    {
        /* holds all menu items */
        public List<MenuItem> allButtons;
      /* holds the index of the clicked menu item*/
       public int clickedButtonIndex;
      /* holds the current keyboard state */
```

```
        public KeyboardState keyboardState;
       /* holds previous keyboard state */
        public KeyboardState oldKeyboardState;
       /* sprite batch to draw menu items */
        public SpriteBatch spriteBatch;
        /* font to render text */
         public SpriteFont spriteFont;

         /* return the clicked button index */
         public int ClickedButtonIndex
         {
              get { return clickedButtonIndex; }
              set
              {
                   clickedButtonIndex = value;
                   if (clickedButtonIndex < 0)
                       clickedButtonIndex = 0;
                   if (clickedButtonIndex >= allButtons.Count)
                       clickedButtonIndex = allButtons.Count - 1;
              }
        }
        public MenuComponent(Game game)
            : base(game)
        {

            allButtons = new List<MenuItem>();
            clickedButtonIndex = 0;
        }
```

3. In the MenuComponent class, add the addMenuItem and CheckKey methods, as shown in the following code. Also, add code in the update and draw methods to update and render each menu item.

```
    //add individual menu item
        public void addMenuItem(string name, SpriteBatch batch,
    SpriteFont font , Vector2 pos)
        {
             spriteBatch = batch;
             spriteFont = font;

             MenuItem item = new MenuItem(name, pos);

             allButtons.Add(item);
        }
       //check whether a key is pressed or not
        private bool CheckKey(Keys theKey)
        {
             return
keyboardState.IsKeyUp(theKey) &&
             oldKeyboardState.IsKeyDown(theKey);
        }
       /*Update the clicked button index as user presses
Up or Down Keys on the keyboard */
        public override void update(GameTime gameTime)
        {
             keyboardState = Keyboard.GetState();
             if (CheckKey(Keys.Down))
             {
```

```
                clickedButtonIndex++;
                if (clickedButtonIndex == allButtons.Count)
                    clickedButtonIndex = 0;
            }
                if (CheckKey(Keys.Up))
            {
                    clickedButtonIndex--;
                if (clickedButtonIndex < 0)
                    clickedButtonIndex = allButtons.Count - 1;
            }
             oldKeyboardState = keyboardState;
           base.Update(gameTime);
        }
/* Draw each menu item with appropriate color
Selected menu item will be highlighted with gold color */
        public override void Draw(GameTime gameTime)
        {
             Color menuColor;
             for (int i = 0; i < allButtons.Count; i++)
             {
                 if (i == clickedButtonIndex)
                     menuColor = allButtons[i].highlight;
                 else
                     menuColor = allButtons[i].normal;
//Draw the menu text on the screen spriteBatch.DrawString
(spriteFont,allButtons[i].menuName, allButtons[i].position,
menuColor);
             }
    }
}
```

4. In your Game class, declare an object of MenuComponent.

```
MenuComponent menuSystem;
```

5. In the LoadContent method, load the required sprite font and add the menu items.

```
protected override void LoadContent()
       {
            /* Create a new SpriteBatch, which can be used to
draw textures.*/
            spriteBatch = new SpriteBatch(GraphicsDevice);
            SpriteFont font = Content.Load<SpriteFont>("Arial");
            string[] menuItems = { "Start Game", "High Scores",
"End Game" };
            Vector2 pos = new Vector2(10.0f,10.0f);
        menuSystem = new MenuComponent(this);
       //Add individual menu item
            for (int i = 0; i < menuItems.Length; i==)
            {
menuSystem.addMenuItem(menuItems[i], spriteBatch, font, pos);
                pos.Y += 30.0f;
            }
       Components.Add(menuSystem);
}
```

6. In your Game.Update method, add the following code:

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

        base.Update(gameTime);
     }
```

7. Override the Draw method as shown in the following code:

```
protected override void Draw(GameTime gameTime)
{
         /* clear the window */
        GraphicsDevice.Clear(Color.CornflowerBlue);
        /*call the Begin method of the SpriteBatch class to
        start drawing the image for the menu.*/

        spriteBatch.Begin();
        base.Draw(gameTime);
        spriteBatch.End();

 }
```

The given example creates three menu items:

- Start Game
- High Scores
- End Game

The code creates a class named `MenuItem` to hold the information of each of the menu item, such as the item name and its position on the screen. The code then creates a `DrawableGameComponent` class type called `MenuComponent`. This `MenuComponent` class performs the following tasks:

- Creates the list of menu items
- Tracks the key press generated from the keyboard and accordingly selects a particular menu
- Draws each menu item and displays the selected menu item in a different color

Figure 5-5 shows the screen with the list of menu items. Notice that the selected menu item is highlighted in the specified color.

Managing Save-Load

Managing save-load involves providing the player with an option for saving and loading the game from a specific point or level.

CERTIFICATION READY
How will you handle game saves at certain important milestones in your FPS game?
3.2

In some instances of a game, a player is not able to complete a level because of the deteriorating health of the game character or the player's failure to complete a task. In such cases, the player can replay the current level with a new inventory. The player might also choose to restart the game or simply give up and exit the game. To enable the player to replay the game from a previous level, you save the game levels along with the user settings. You should provide a UI that the player can use to save and/or load the game from a specific point or level.

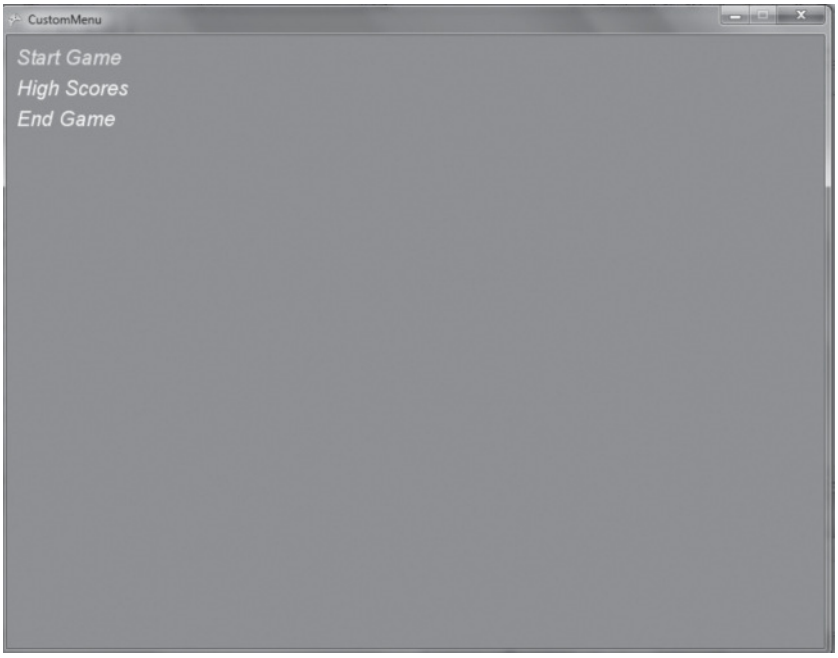


Figure 5-5

The game screen showing the menu options

PROGRAMMING SAVE-LOAD UI

Consider an FPS game in which the player character has to diffuse a bomb using a bomb disposal kit. To achieve this objective, the character has to kill all the enemies and find the disposal kit. Given that it is a difficult and time consuming mission, you can program the game to provide the flexibility of saving the game state at certain important milestones. This helps retain the player's interest in the game and ensures that the player's efforts are not futile and the mission becomes doable. To facilitate this feature, you can provide the ***save-load UI*** for the player. The following code creates such a UI for the purpose of save-load. The code creates a save button and loads a button user interface by creating the following helper classes:

- `XNAButton` class to hold the save button and load button information
- `MouseCursor` class to handle mouse states

The code assumes that a project named `SaveLoadInterface` is already created.

The following is the `XNAButton` class. This class creates a button to provide an interface for the save and load option. The class also creates a click event handler for the button's click event. An event is a means of helping a class provide notifications to users of that class to indicate that something has happened to an object. Event handlers are functions or methods that are executed whenever an event such as a button click is triggered.

```
namespace SaveLoadInterface
{
    class XNAButton
    {
      /* sprite batch to draw the button */
        public SpriteBatch spriteBatch;
      /* sprite font to draw the text on the button
*/
        public SpriteFont   spriteFont;
      /*A button usually has two states: Normal and Clicked. The code
uses two different textures to differentiate between these two states. */

       public Texture2D normalTexture, highlightTexture;
     /*Bounding rectangle to determine the area of the button. It is
important to know whether button is active or not. The code achieves
```

```
this by checking whether the mouse cursor position is present inside
the rectangle or not. */

        public Rectangle boundingRect;
/* Screen position to draw the button. */

        public Vector2 position;
/* a string variable to hold the button Name. */
        public string buttonName;
/* define a ButtonState structure to hold the button state information */

        public ButtonState previousButtonState;
/* define an event handler to execute an action when the button is
pressed.*/

        public event EventHandler onButtonPressed;
       /* constructor to initialize button */
        public XNAButton(Texture2D normalTexture, Texture2D
highlightTexture , SpriteBatch spriteBatchRef ,SpriteFont font ,
Vector2 pos , string name)
        {
             this.normalTexture = normalTexture;
             this.highlightTexture = highlightTexture;
             this.spriteBatch = spriteBatchRef;
             this.spriteFont = font;
             this.position = pos;
             this.buttonName = name;

             boundingRect = new Rectangle((int)pos.X,(int) pos.Y,
normalTexture.Width, normalTexture.Height);
             previousButtonState = ButtonState.Released;
        }

        public void Draw(GameTime gameTime)
        {
            /* get mouse point by using the MouseState object as it
holds the position information of the mouse cursor. */

             MouseState mouseState = Mouse.GetState();
             Vector2 mousePt = new Vector2(mouseState.X,
mouseState.Y);

             spriteBatch.Begin();

             /* Check if mouse is inside button bounding rectangle */
             if ((mousePt.X > boundingRect.Left && mousePt.X <
boundingRect.Right) &&
                  (mousePt.Y > boundingRect.Top &&
mousePt.Y < boundingRect.Bottom))
             {
                 //Check if button is clicked
                 if (mouseState.LeftButton == ButtonState.Pressed)
                 {
                     /*If button is clicked, render it with HighLight
texture */

spriteBatch.Draw(highlightTexture, position, Color.White);
/* save the buttonstate as previousButtonState which is used to track
when button is released */

                     previousButtonState = ButtonState.Pressed;
```

```
                }
                else
                {
                    spriteBatch.Draw(normalTexture, position, Color.
White);
                }
//Check if button is released or not .If button is released , trigger
the event by calling EventHandler object */
                if (mouseState.LeftButton == ButtonState.Released &&
previousButtonState == ButtonState.Pressed)
                {
                    /*If button pressed execute the Event for that
button */
                    if (onButtonPressed != null)
                    {
                        onButtonPressed(this, null);
                    }
             /* save the state of the button */
                    previousButtonState =
ButtonState.Released;
                }
             }
             else
             {
                spriteBatch.Draw(normalTexture, position, Color.
White);
             }
             /* Render button text. The code checks the length of the
string i.e. the button name and align it according to the button
position and size of the button texture. */
             Vector2 stringSize =
spriteFont.MeasureString(buttonName);
             Vector2 textPosition = new
Vector2(position.X + normalTexture.Width / 2.0f - stringSize.X/2.0f,
position.Y + normalTexture.Height / 2.0f - stringSize.Y/2.0f);
             spriteBatch.DrawString(spriteFont, buttonName,
textPosition, Color.Black);
             spriteBatch.End();
         }
    }
}
```

Now that you have created a button interface for the save and load option, look at the code to create a helper class to render a mouse cursor.

```
namespace SaveLoadInterface
{
        public class MouseCursor
        {
          /* mouse cursor texture */
            Texture2D mouseCursor;
          /* sprite batch to draw the mouse cursor texture */
            SpriteBatch spriteBatch;
            public MouseCursor(Texture2D texture, SpriteBatch
spriteBatchRef)
        {
            mouseCursor = texture;
```

```
            spriteBatch = spriteBatchRef;
        }

        public void Draw(GameTime gameTime)
        {
            /* draw the mouse cursor texture at mouse position */
            MouseState ms = Mouse.GetState();
            Vector2 mousePos = new Vector2(ms.X, ms.Y);

            spriteBatch.Begin();
            spriteBatch.Draw(mouseCursor, mousePos, Color.White);
            spriteBatch.End();
        }
    }
}
```

Now add the following code to your Game class:

```
namespace SaveLoadInterface
{
        public class Game1 : Microsoft.Xna.Framework.Game
        {
            GraphicsDeviceManager graphics;
            SpriteBatch spriteBatch;
          /* create an object of MouseCursor to handle mouse movement */
            MouseCursor mouseCursor;
         /* create save and load button */
            XNAButton saveButton, loadButton;

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

            protected override void LoadContent()
            {
                // Create a new SpriteBatch, which can be used to
draw textures.
                spriteBatch = new SpriteBatch(GraphicsDevice);
                /* load normal texture for buttons */
                Texture2D normalTexture = Content.Load<Texture2D>
(@“Normal”);
                /* load highlight texture for buttons */
                Texture2D highlightTexture = Content.Load<Texture2D>
(@“Highlite”);
                /* load sprite font to draw button name*/
                SpriteFont font = Content.Load<SpriteFont>
(@“Arial”);

                //Initialize Load Button
                loadButton = new XNAButton(normalTexture,
highlightTexture, spriteBatch, font,
                                           new Vector2(10.0f, 10.0f),
“LOAD”);

                //Initialize Save Button
                saveButton = new XNAButton(normalTexture,
highlightTexture, spriteBatch, font,
                                           new Vector2(10.0f, 20.0f +
normalTexture.Height), “SAVE”);
```

```
                /* Add event handler to load button and save button.
These event handlers will be executed whenever the button is clicked */
                loadButton.onButtonPressed += new EventHandler
(loadSelected);
                saveButton.onButtonPressed += new EventHandler
(saveSelected);

                //initialize Mouse cursor
                Texture2D mouseCursorTexture = Content.Load<Texture2D>
(@"Arrow");
                mouseCursor = new MouseCursor(mouseCursorTexture,
spriteBatch);
            }
/* define the event handler method for the save button */

            private void saveSelected(object sender, EventArgs e)
            {
                //Save your Game here
            }
/* define the event handler method for the load button */
            private void loadSelected(object sender, EventArgs e)
            {
                //Load your Game here
            }

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

             base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {

GraphicsDevice.Clear(Color.CornflowerBlue);

             //Draw the buttons

             saveButton.Draw(gameTime);
             loadButton.Draw(gameTime);

             mouseCursor.Draw(gameTime);

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

When you run your code, you might see two buttons appearing on the screen along with the mouse cursor (see Figure 5-6). One button is for save and the other button is for load. On clicking either of these two buttons, you notice that other than the change in the color of the buttons or a display message, no saving or loading happens. This is because you have not written the actual code for saving and loading the game. Figure 5-7 shows the game screen displaying a message and shows the Load button in the highlighted color. This indicates that the player has pressed the button.

Figure 5-6

The game screen displaying the save-load interface

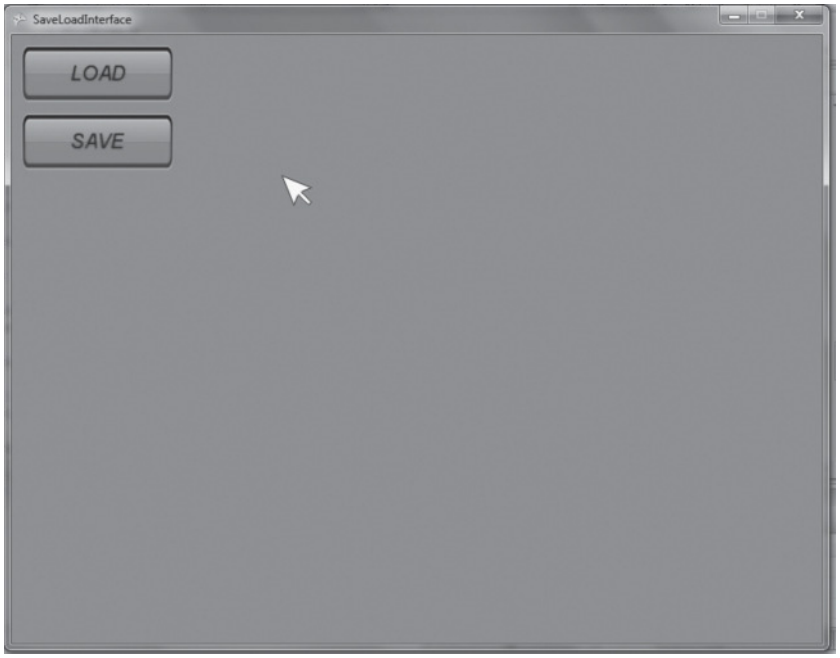
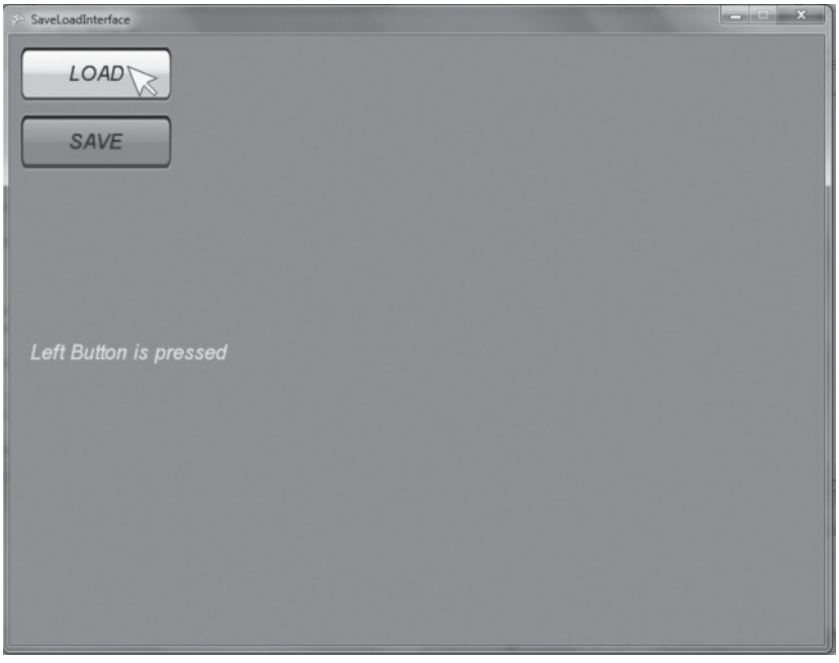


Figure 5-7

The game screen displaying the status of the button press



CERTIFICATION READY
How will you program the UI for managing game saves and game loads in XNA 4.0?
1.7

■ Programming the UI Game States

THE BOTTOM LINE

Programming UI game states involves mapping the UI with a specified state and programmatically deciding the behavior of the UI based on the current state.

It is important to program how the UI elements need to behave at different game states in the game. As already discussed in Lesson 4, "Designing Specific Game Components," you can

CERTIFICATION READY
How will you control the behavior of the UI using game states?
1.4

map each UI element to a particular game state. This helps you to decide the behavior of the specified UI element at a given state.

Defining UI Behavior Using States

Defining the various valid states for your UI objects helps your game decide which action to perform when in that state.

For example, assume that you have defined a state called *Display Menu* in your game and have associated the specified menu asset with this state. When your game enters this state, you can display the menu asset to the player. This way you can define the behavior of the menu asset in this particular state.

You will recall that you can declare a variable to store the value of the game state at any point in the game. The game state variable can assume Boolean values or custom-enumerated values. The game state variable is instrumental in managing the game states.

Once you define the enumeration, you can use the game state variable to hold any value from those defined in the enumeration. How the UI assets will behave in different game states can then be accordingly defined for each game state. The code to program UI game states uses a simple case-loop to determine the current game state and direct the UI assets to behave according to the properties defined for the particular game state.

To get a better understanding of how to track the UI asset programmatically using game states, let us look at a simple example on how to display different screens—a Menu screen that displays a simple menu list, a GameRule screen that displays the rule of the game, and a Title screen that displays the title of the game. The steps and the associated code snippets illustrate how the UI game states are managed.

DISPLAY SCREENS USING STATE

GET READY. Create your project and name it `DisplayScreenUsingState`.

1. Define a class `Game1` by inheriting from `Game`.

```
public class Game1 : Microsoft.Xna.Framework.Game
```

2. Add the following code in the constructor:

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

3. Define class-level objects.

```
/*Background textures for the various screens in the
game */
    Texture2D mMenuScreenBackground;
    Texture2D mTitleScreenBackground;
    Texture2D mGameRuleScreenBackground;
    /*The enumeration of the three screen states available
in the game*/
```

```
enum ScreenState
{
    Menu,
    Title,
    GameRule
}
//The current screen state
ScreenState mCurrentScreen;
```

4. Override the `LoadContent` method, as shown in the following code:

```
protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw
textures.
        spriteBatch = new
SpriteBatch(GraphicsDevice);
        //Load the screen backgrounds
        mMenuScreenBackground = Content.Load<Texture2D>
("MenuScreen");
        mTitleScreenBackground = Content.Load<Texture2D>
("TitleScreen");
mGameRuleScreenBackground = Content.Load<Texture2D>
("GameRuleScreen");
/*Initialize the current screen state to the screen that you
want to display first*/
        mCurrentScreen = ScreenState.Menu;
    }
```

5. Override the `Update` method, as shown in the following code:

```
protected override void Update(GameTime gameTime)
    {
        // Allows the game to exit
        if
(GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed)
            this.Exit();
        /*Call the method associated with the current screen*/
        switch (mCurrentScreen)
        {
            case ScreenState.Menu:
                {
                    MenuScreen();
                    break;
                }
            case ScreenState.Title:
                {
```

```
                TitleScreen();
                break;
            }
        case ScreenState.GameRule:
        {
            GameRuleScreen();
            break;
        }        }

            base.Update(gameTime);
    }
```

6. Add the `MenuScreen` method, as shown in the following code:

```
    private void MenuScreen()
    {
        /* check whether the user has pressed the key T. If yes,
make a transition to title screen
        */
        if (Keyboard.GetState().IsKeyDown(Keys.T) == true)
            {
                mCurrentScreen = ScreenState.Title;
            }
              /* check whether the user has pressed the key R. If
yes, make a transition to game rule screen
              */
            else if(Keyboard.GetState().IsKeyDown(Keys.R) == true)
        {
            mCurrentScreen = ScreenState.GameRule;
        }
    }
```

7. Add the `TitleScreen` method, as shown in the following code:

```
private void TitleScreen()
     {
         /* check whether the user has pressed the key M.
If yes, make a transition to title screen
         */
     if (Keyboard.GetState().IsKeyDown(Keys.M) == true)
     {
         mCurrentScreen = ScreenState.Menu;
     }
         /* check whether the user has pressed the
key R. If yes, make a transition to game rule screen
         */
```

```
      else
if(Keyboard.GetState().IsKeyDown(Keys.R) == true)
      {
         mCurrentScreen = ScreenState.GameRule;
      }

}
```

8. Add the GameRuleScreen method, as shown in the following code:

```
private void GameRuleScreen()
      {
         /* check whether the user has pressed the key T.
If yes, make a transition to title screen
         */
      if (Keyboard.GetState().IsKeyDown(Keys.T) == true)
      {
         mCurrentScreen = ScreenState.Title;
      }
            /* check whether the user has pressed the key M.
If yes, make a transition to game rule screen
            */

      else
if(Keyboard.GetState().IsKeyDown(Keys.M) == true)
           {
              mCurrentScreen = ScreenState.Menu;
           }
      }
```

9. Override the Draw method, as shown in the following code:

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

      GraphicsDevice.Clear(Color.CornflowerBlue);

            // add your drawing code here
            spriteBatch.Begin();

            // call the Draw method associated with the
current screen
            switch (mCurrentScreen)
            {
               case ScreenState.Menu:
                  {
                     DrawMenuScreen();
                     break;
                  }
               case ScreenState.Title:
```

```
                {
                    DrawTitleScreen();
                    break;
                }
            case ScreenState.GameRule:
                {
                    DrawGameRuleScreen();
                    break;
                }
        }

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

10. Add the DrawMenuScreen method, as shown in the following code:

```
private void DrawMenuScreen()
        {
            /*draw all of the elements that are part of the Menu
screen*/
            spriteBatch.Draw(mMenuScreenBackground, Vector2.Zero,
Color.White);
    }
```

11. Add the DrawTitleScreen method, as shown in the following code:

```
private void DrawTitleScreen()
        {
            /*draw all of the elements that are part of the
title screen*/
            spriteBatch.Draw(mTitleScreenBackground, Vector2.
Zero, Color.White);
        }
```

12. Add the DrawGameRuleScreen method, as shown in the following code:

```
private void DrawGameRuleScreen()
        {
            /*draw all of the elements that are part of the game
rule screen*/
spriteBatch.Draw(mGameRuleScreenBackground, Vector2.Zero,
Color.White);
    }
```

CERTIFICATION READY
What is the mechanism to track UI asset using game states in XNA 4.0?
1.7

The given code maintains a state variable that holds the current screen to be displayed. By using this technique, you can tell your game to display the current screen and maintain a smooth screen transition. When you run the game, the game should greet you with the menu screen. When the user presses the "T" or "R" key, he can see the title screen or the game rule screen based on his key press.

■ Programming the UI Access Mechanisms

THE BOTTOM LINE Programming ***UI access mechanisms*** involve creating the required GUI controls for your game.

CERTIFICATION READY
How are UI access mechanisms helpful in games?
1.4

UI access mechanisms help the players to interact with the game. For example, a player can save or load a game with the click of a button or select the required tools for his inventory by selecting the options given in a check box. You can see that GUI controls have become an integral part of game programming.

You will probably have various kinds of GUI controls or UI assets in your game. These can include:

- Button
- Label
- Textbox
- Check box
- Radio button
- Picture box
- Form

Just as you have different controls, you also have the corresponding event handlers for these controls. The different event handlers might be:

- Mouse click
- Mouse enter
- Mouse leave
- Mouse move
- Mouse down
- Toggle
- Close

Programming the UI Control

Programming the UI control involves creating a UI access mechanism control through code.

CERTIFICATION READY
How will you program the UI access mechanism for your game in XNA 4.0?
1.7

Let us look at the tasks involved in creating a UI access mechanism control for your game. For example, suppose you want the player to select from a list of images displayed using a check box control. For this, you need to:

- Create a check box control.
- Create the method that contains the code to be executed when the check box is selected.
- Create an `OnClick` event handler that maps the check box control with its corresponding method.

You can create a UI control in your game either by using `System.Windows.Forms` or by using a third-party library in combination with the XNA 4.0 built-in classes. You can also create the UI controls using XNA Sprite.

TAKE NOTE* Although you can use `System.Windows.Form` to create controls, it limits the freedom of using UIs that match the required UI theme of the game. Each game has its own UI theme and the UI should follow that theme.

The following sample code creates a custom check box using XNA Sprite. The code assumes that a project named XNACheckbox has already been created.

```
namespace XNACheckbox
{
    public class CheckBox
    {
      //check box position
    public int mXPosition;
    public int mYPosition;
       /* texture for check box. The code uses two different
textures for showing checked and unchecked checkbox .*/

    protected Texture2D mUncheckedImage;
    protected Texture2D mCheckedImage;

    //mouse states to handle click
    protected MouseState mCurrentState;
    protected MouseState mPreviousState;
    protected Rectangle mHotSpot;
    protected SpriteFont mSpriteFont;
    protected Vector2 mSpriteFontLocation;

    //delegate to handle Mouse click
    public delegate void OnCheckBoxMouseClick();
    public OnCheckBoxMouseClick OnMouseClick;

    //variable to store state of the checkbox
    public bool mbChecked;
    public Color mSpriteFontColor;

    //check box text
    public string  mText;

    public CheckBox(Texture2D uncheckTexture, Texture2D
checkTexture, SpriteFont spriteFont,
          Color color, string txt, int xpos, int ypos)
    {
          mUncheckedImage = uncheckTexture;
          mCheckedImage = checkTexture;
          mText = txt;
          mXPosition = xpos;
          mYPosition = ypos;
          mSpriteFont = spriteFont;
          mHotSpot = new Rectangle(xpos, ypos, uncheckTexture.
Width, uncheckTexture.Height);
          mSpriteFontLocation = new Vector2(mHotSpot.Right + 1,
mHotSpot.Y);
          mbChecked = false;
          mSpriteFontColor = color;
    }

       public void Draw(SpriteBatch spriteBatch)
       {
           /* highlight the box as soon as mouse touches it */

            if (this.mbChecked == true)
            {
```

```
                if (mHotSpot.Contains(new
Point(mCurrentState.X, mCurrentState.Y)))
                {
                    spriteBatch.Draw(mCheckedImage,
                        mHotSpot,
                        Color.Gold);
                    spriteBatch.DrawString(mSpriteFont,
mText, mSpriteFontLocation, mSpriteFontColor);
                }
                else
                {
                    spriteBatch.Draw(mCheckedImage,
                        mHotSpot,
                        Color.Gray);
                    spriteBatch.DrawString(mSpriteFont, mText,
mSpriteFontLocation, mSpriteFontColor);
                }
            }
            else
            {
                if (mHotSpot.Contains(new Point(mCurrentState.X,
mCurrentState.Y)))
                {
                    spriteBatch.Draw(mUncheckedImage,
                        mHotSpot,
                        Color.Gold);
                    spriteBatch.DrawString(mSpriteFont, mText,
mSpriteFontLocation, mSpriteFontColor);
                }
                else
                {
                    spriteBatch.Draw(mUncheckedImage,
                        mHotSpot,
                        Color.Gray);
                    spriteBatch.DrawString(mSpriteFont, mText,
mSpriteFontLocation, mSpriteFontColor);
                }
            }
        } // end draw

        public void update()
        {

             mCurrentState = Mouse.GetState();
            //Check if mouse is clicked or not
             if (mCurrentState.LeftButton == ButtonState.
Pressed &&
                mPreviousState.LeftButton == ButtonState.
Released)
            {
                /* Check if user clicked the mouse inside the
check box */

                if
(this.mHotSpot.Contains(mPreviousState.X,
mPreviousState.Y))
```

```
                {
                /* If user clicked inside the check box then
call the delegate OnMouseClick().*/
                    try
                    {
                        this.OnMouseClick();
                    }
                    catch { }
                    this.mbChecked = !this.mbChecked;
                }
            }
            mPreviousState = mCurrentState;
        }
    }
}
```

In your `Game` class, declare an object of the CheckBox type, as shown below.

```
CheckBox customCheckBox;
```

In your `Game.LoadContent()` write the following code.

```
protected override void LoadContent()
        {
            /* create a new SpriteBatch, which can be used to
draw textures */
            spriteBatch = new SpriteBatch(GraphicsDevice);
            /* load your checked and unchecked image */
            Texture2D checkedImage = Content.Load
<Texture2D>(@"your image1");
            Texture2D unCheckedImage = Content.Load<Texture2D>
(@"your image2");
            SpriteFont font = Content.Load<SpriteFont>
(@"Arial");
/* assign the delegate to be called when user clicked inside
the check box. */
            customCheckBox = new CheckBox(unCheckedImage,
checkedImage, font, Color.Wheat, "My checkBox", 10, 20);
            customCheckBox.OnMouseClick = new CheckBox.OnCheckB
oxMouseClick(mouseDelegate);
}
```

Define the mouse delegate, as shown in the following code:

```
public void mouseDelegate()
{
       /* Write the code to perform the specific action when
the player checks the checkbox. */
}
```

In your `Game.Update` loop, update the check box state, as shown in the following code:

```
customCheckBox.update();
```

Write the following code in your `Game.Draw` to render the check box:

```
spriteBatch.Begin();
customCheckBox.Draw(spriteBatch);
spriteBatch.End();
```

When you run the code, you see the game screen displaying the custom check box (see Figure 5-8). When you place the mouse cursor on the check box, you see that the check box is displayed in a golden color (see Figure 5-9). Once you click on the check box, you can see the check box as selected (see Figure 5-10).

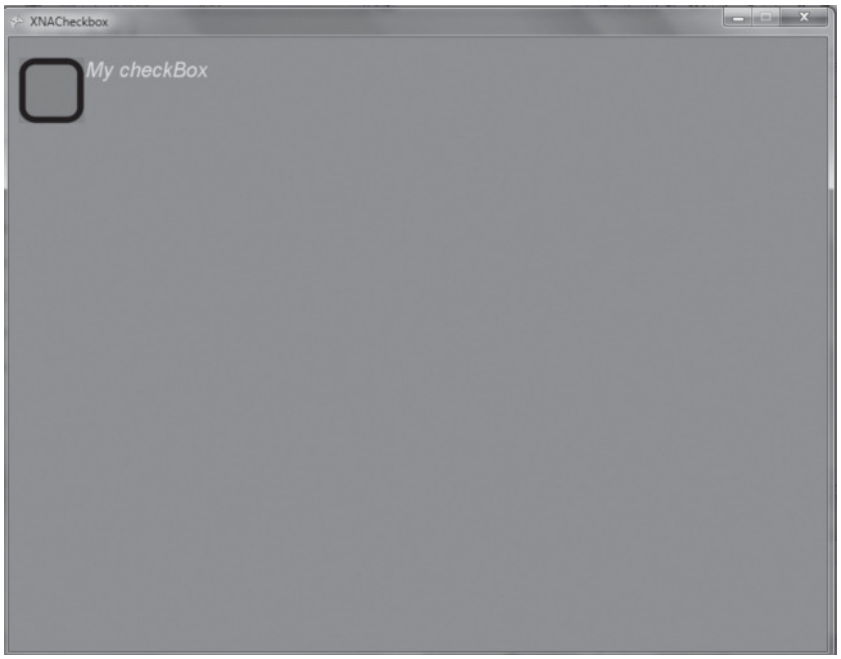


Figure 5-8

The game screen displaying the custom check box

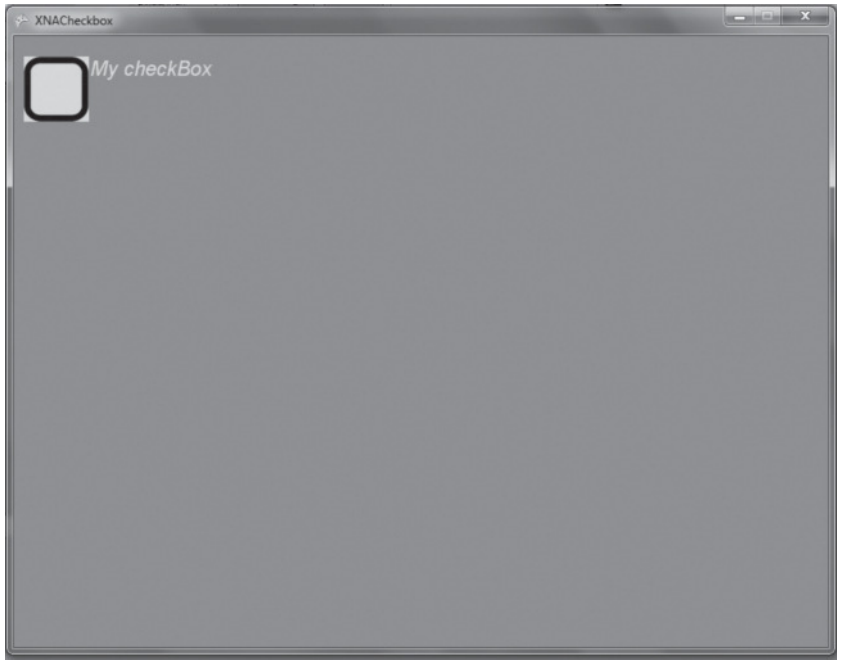
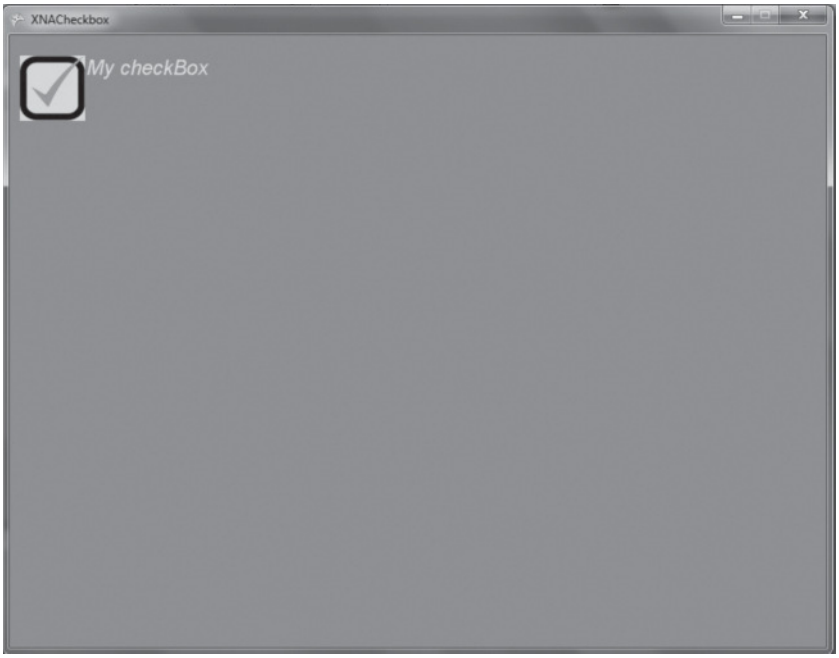


Figure 5-9

The game screen showing the check box in a golden color

Figure 5-10

The game screen showing the check box as checked



SKILL SUMMARY

In this lesson, you learned:

- Managing UI assets involves creating and making available the different UI assets for the game.
- Loading UI assets involves adding the set of instructions to the game code to create or load the UI assets and make them available for use in the game as envisaged during the game.
- Configuring the audio, video, and player input with the UI assets involves loading the audio and video file for the respective UI assets and mapping the keyboard, mouse, and control pad inputs with the respective UI assets.
- You might need to retrieve the user input data to perform certain actions on the game screen. For such instances, you can capture the state of an input device using the classes and structures available in the `Microsoft.Xna.Framework.Input` namespace.
- Menus are an integral part of the UI of every multimedia application including games. As you already learned in Lesson 3, you create menus in your game to provide the player with a list of options.
- Save-load refers to the process of saving the game and loading the saved game. You can create the UI for a save-load game by creating buttons using sprites.
- Programming UI game states involves mapping the UI with a specified state and deciding on the behavior of the UI based on the current state programmatically.
- Programming UI access mechanisms involves creating the different UI access controls, such as button, label, and check box, and linking these UI assets with the respective functionality.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ provides you with the freedom of creating UIs that match the required UI theme of the game.
2. _____ class helps to incorporate background sound effects for a game's UI asset.
3. Mapping your UI with the respective _____ helps you to decide the behavior of the UI at designated areas in a game.
4. _____ UI lets the player save and load the game.
5. _____ class represents the video being displayed on the screen.
6. _____ method helps you load the required UI assets in your game.
7. A(n) _____ class in XNA 4.0 provides a modular approach to adding graphics content to your game.
8. The classes and structures in the _____ namespace help you to retrieve user input from different input devices.
9. The XNA Framework 4.0 provides you with the _____ that helps you to access your game assets directly in your XNA Game Studio projects.
10. You need to load the UI assets of your game to the game _____ project to make them available to the content pipeline.

Multiple Choice

1. You decide to play a video on a button click on the start-up screen of your game. The video displays different characters narrating their role in the gameplay. Which of the following classes and methods would help you load and play the video? (Choose all that apply.)
   a. `Video`
   b. `VideoPlayer`
   c. `Content.Load()`
   d. `SoundEffect`

2. You decide to add graphics resources to your game. Which of the following methods would help you load the resources to the game screen?
   a. `Game.LoadContent`
   b. `Game.Draw`
   c. `SpriteBatch.Draw`
   d. `SpriteBatch.Begin`

3. You decide to create menus using the drawable game component. Which of the following methods of your drawable game component class will be automatically called from your Game class once you register the component with your game? (Choose all that apply.)
   a. Initialize method
   b. Update method
   c. Draw method
   d. Constructor of your drawable game component class

4. Which of the following classes and structures help you detect the current state of a mouse input device? (Choose all that apply.)
   a. Keyboard
   b. Mouse
   c. Gamepad
   d. `MouseState`
5. Which of the following namespace does the `VideoPlayer` class belong to?
   a. `Microsoft.Xna.Framework.Media`
   b. `Microsoft.Xna.Framework.Input`
   c. `Microsoft.Xna.Framework.Content`
   d. `Microsoft.Xna.Framework.Audio`
6. Which of the following namespaces would you include in your project to capture the input states of the various input devices?
   a. `Microsoft.Xna.Framework.Media`
   b. `Microsoft.Xna.Framework.Input`
   c. `Microsoft.Xna.Framework.Content`
   d. `Microsoft.Xna.Framework.Audio`
7. You decide to include a user interface for save-load. Which of the following UI controls would best suit your purpose?
   a. Button
   b. Check box
   c. Menu
   d. Radio button
8. You have multiple screens in your game. Which of the following options would help you track screen transitions?
   a. Save load UI
   b. Game state variable
   c. Menu
   d. Content pipeline
9. Which of the following methods will register the drawable game component with your game class?
   a. `Load.content` method
   b. `Game.Components.Add` method
   c. `Game.Update` method
   d. `Game.Draw` method
10. You decide to use a menu in your game. For which of the following options can you use menu user interface? (Choose all that apply.)
    a. Game story
    b. Game space
    c. Non-diegetic component
    d. Meta component

■ Competency Assessment

Scenario 5-1: Adding Sound

You are developing an adventure game. The gameplay involves the player character unlocking different doors to reach a treasure. You want to include a background sound whenever a door is unlocked. Write the code to include the required sound in your game.

Scenario 5-2: Capturing Key Press

You develop an FPS game. In a particular scene, you need to capture the key press "A" to fire bullets from a gun. How would you capture the key presses generated from a keyboard in XNA 4.0?

■ Proficiency Assessment

Scenario 5-3: Creating a Menu

You are developing a board game. The opening screen of your game should provide various options to the player. The screen should provide a user interface to let the player start the game, view achievements, view options available in the game, and close the game.

1. How would you recommend solving this problem?
2. How would you programmatically create the required user interface?

Scenario 5-4: Tracking the UI Using States

In your game, you display a menu asset for a particular level. The menu asset lists the available weapons in that level. You need to track the display of the menu asset during the gameplay.

1. How would you recommend this problem be solved?
2. How would you programmatically achieve the tracking of the menu display?

Developing the Game Functionality

LESSON 6

EXAM OBJECTIVE MATRIX

SKILLS/CONCEPTS	MTA EXAM OBJECTIVE	MTA EXAM OBJECTIVE NUMBER
Programming the Components	Understand Components. Capture User Data. Work with XNA.	1.5 1.6 1.7
Handling Game Data	Work with XNA.	1.7

KEY TERMS

breadcrumb path following
deserialization
game data
influence map
save game file
terrain analysis
tool creation
visibility graph
waypoint navigation

Steve Watson works at FunGaming Inc., and leads the development team. His company is developing a first-person shooter (FPS) game for console and computer systems. The team members are in the development phase. They have developed the code to load their game's user interface (UI) assets, such as menus, UI controls, videos, game models, and so on in a specific sequence as per the gameflow. They are now in the process of making their game fully functional.

A game comes alive when each game component is made functional in an integrated fashion as per the design. In this development phase, the team members decide to program the behavior of individual game components so that they look and behave like a part of the game story. They also decide to build artificial intelligence (AI) in their game characters. To ensure that player preferences and progress in the game are tracked, Steve's team decides to capture user data. They also programmatically manage the flow of their game using game states.

■ Programming the Components

THE BOTTOM LINE

In the simplest terms, functionality in a game refers to the behavior of the different components of the game. Players enjoy a game when they find the game challenging and, at the same time, easy to play. You need to carefully balance the functionality of each element in your game such that the players find the game to be an enjoyable experience.

Players generally have a choice of options to perform in a game. They can choose their weapons or tools to overcome obstacles in the game. They can achieve certain rewards for completing a task or level in the game. However, to complete the game, the players should have access to the status of the game components. For example, when a player uses a gun to shoot the opponent character, you should program the game to decrease the number of bullets in the gun. It is equally important to inform the player of the number of bullets left in the gun, maybe through a heads up display (HUD). Similarly, when a player character is hit, you should program your game to show a decrease in the health of the player character. You can do this by making your game display a HUD that will show the decrease in player character's health. Additionally, you can even program AI into the game character to serve as an intelligent opponent to the player.

A significant part of game development goes into creating and programming the game functionality. The game development process comprises the following elements:

- Creating tools
- Programming the game
- Incorporating AI

Understanding Tool Creation

The game development process also involves creating game tools to handle game specific tasks.

CERTIFICATION READY
What is game tool creation?
1.5

Almost every game involves the use of game tools for tasks such as importing or converting art, building levels, and so on. These tools facilitate the development of video games and are created by game tool programmers. ***Tool creation*** is the process through which a game programmer creates game tools. For example, a tool programmer might be required to create asset conversion tools for game projects. Asset conversion tools are programs that convert artwork, such as 3D models, into formats required by the game. Although the standard 3D modeling software, such as 3D Studio Max or Autodesk Maya, include importers and exporters to handle these tasks, games might use other custom graphics or image file formats. The tool programmer might then be required to create a custom asset conversion tool specific to a game.

Certain other tasks of a tool programmer include creating prototypes, level editors, or map editors. Level editors or map editors are software tools that might be released along with the game. The level editor tool helps the game player customize or modify levels within a game. A map editor tool allows the game player create maps without programming skills. For example, using a map editor, the game player can use the suite of objects, weapons, and other components included in the game in the method of his choice. Modern games use several tools for assistance in the game development process.

Programming the Game

Programming the game involves incorporating the required functionality in the specific game components through code.

CERTIFICATION READY
How will you program a speedometer to show the speed in a racing game?
1.5

After you load or add game components such as sprites, textures, game models, and images to your game, you need to make the components functional. However, to make the components functional, as per the player's input or as per the game's progress, you need to further program the components.

For example, in a shooting game, your game components might include a gun, bullets, and HUD objects. As already discussed, when the player in your game fires a bullet, the HUD showing the number of bullets must show the decreased number of available bullets in the player character's gun. Similarly, when a bullet from an opponent hits the player, the health status of the player character must be shown to decrease using a health bar. Such concurrent updates as per the game's progress are what you need to program while programming the components.

ADDING FUNCTIONALITY

Let us reconsider the example of the health bar component. For the health bar to be functional, you have to add the necessary code based on the player character's health. For example, if the player character's health deteriorates, you have to program the health bar component to show the decrease in health, and if the player character's health improves, you have to program the health bar to show the improvement.

The following code sample adds a health bar using an XNA sprite and modifies the health bar based on the up and down key presses. As discussed in Lesson 3, "Creating the Game Output Design," sprites are 2D images. You can create sprites using any drawing program, such as Adobe Photoshop, or even using Microsoft Paint.

TAKE NOTE*
Texture 2D objects store images.

Once you create sprites, you can render sprites by loading them in the Texture2D objects. You can then draw the sprites using the SpriteBatch class.

The code assumes that a project named "XNAPlayerHealth" is already created. The code uses the content member to load the health bar texture from the XNA Framework Content Pipeline. This example assumes that the texture named "playerhealth" is available in the project.

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

        //Texture for your health bar
        Texture2D playerHealthBar;
        // Current player health set to 100
        int playerCurrentHealth = 100;

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

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

            //Load your player health texture
            playerHealthBar = Content.Load<Texture2D>
(@"playerHealth");
        }

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

              KeyboardState keyState = Keyboard.GetState();

              //If the Up Arrow is pressed, increase the Health bar
              if (keyState.IsKeyDown(Keys.Up) == true)
              {
                  playerCurrentHealth += 1;
              }

              //If the Down Arrow is pressed, decrease the Health bar
              if (keyState.IsKeyDown(Keys.Down) == true)
              {
                  playerCurrentHealth -= 1;
              }

              /* Clamp the health between 0 and 100 using
MathHelper.Clamp method. The MathHelper.Clamp method restricts
a value to be within a specific range. In this case, the method
will restrict the value of the playerCurrentHealth variable
between 0 and 100. */

              playerCurrentHealth = (int)MathHelper.
Clamp(playerCurrentHealth, 0, 100);

              base.Update(gameTime);
          }

          protected override void Draw(GameTime gameTime)
          {

        GraphicsDevice.Clear(Color.CornflowerBlue);
              spriteBatch.Begin();

          // Y position for the sprite
                int postionY = this.Window.ClientBounds.Height -
playerHealthBar.Height - 20;

          //X position for Sprite to render
          int positionX = this.Window.ClientBounds.Width / 2 -
          playerHealthBar.Width / 2;

                /* Draw the empty health i.e. amount of health
decreased */
                spriteBatch.Draw(playerHealthBar, new
Rectangle(positionX,
```

```
                    postionY, playerHealthBar.Width, 44),
                    new Rectangle(0, postionY, playerHealthBar.
Width, 44), Color.Gray);

//Needed to show a slow transition from Green to Red
//When health is full , it shows Green Color
//As health decreases , it will slowly turn into Red
float healthPercentage = (float)playerCurrentHealth / 100.0f;

       Color healthBarColor = new Color( 1.0f - healthPercent-
age ,healthPercentage, 0.0f);

                /* Draw the current health level based on the
        current Health */
                spriteBatch.Draw(playerHealthBar, new
        Rectangle(positionX,
                    postionY, (int)(playerHealthBar.Width *
                    ((double)playerCurrentHealth / 100)), 44),
                    new Rectangle(0, postionY, playerHealthBar.
                    Width, 44), healthBarColor);

                    /* Draw the box around the health bar */
                    spriteBatch.Draw(playerHealthBar, new
        Rectangle(positionX,
                      postionY, playerHealthBar.Width, 44),
                      new Rectangle(0, 0, playerHealthBar.Width,
        44), Color.White);

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

When you run the code, you notice that on pressing the up key, the health bar will show an increase in health (see Figure 6-1); and on pressing the down key, the health bar will show a decrease in health (see Figure 6-2).

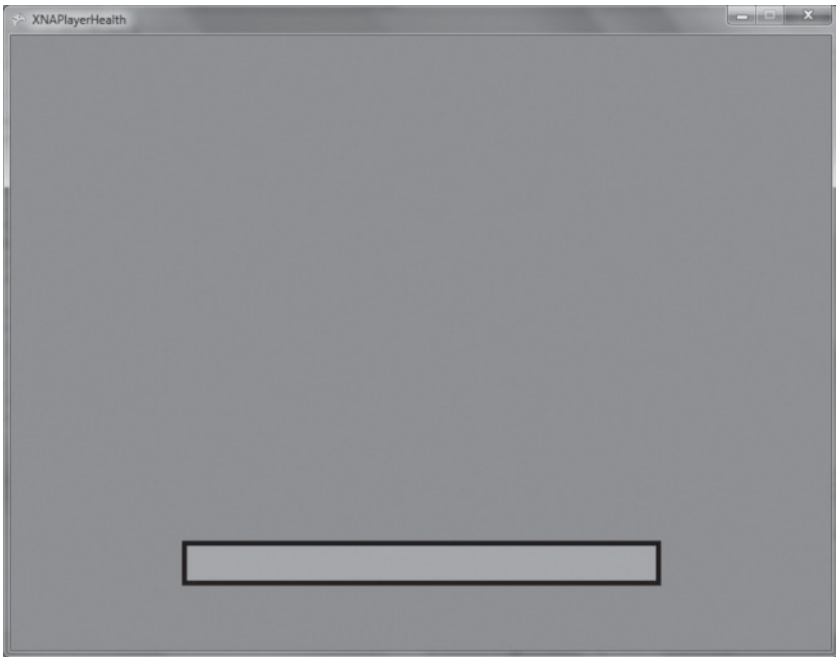


Figure 6-1

The health bar displaying full health

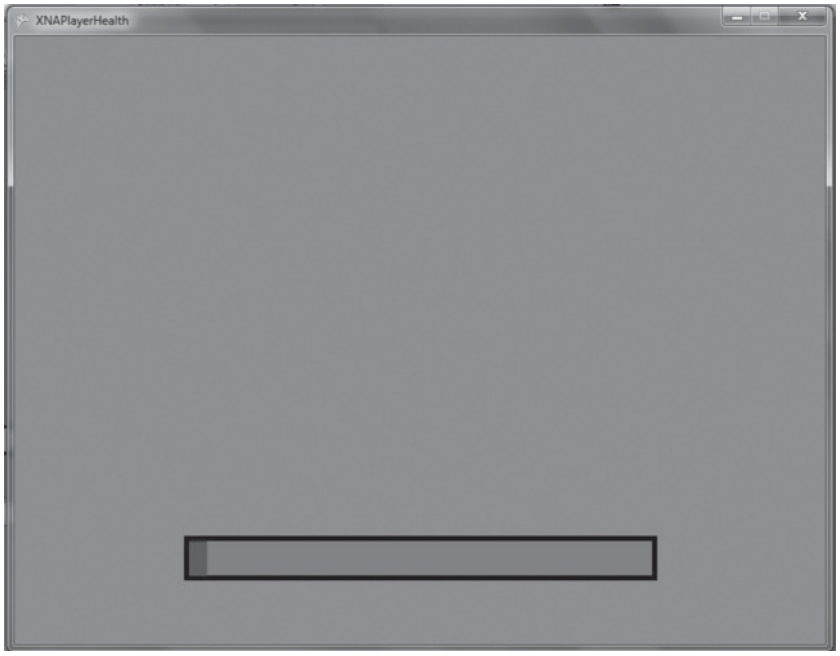


Figure 6-2

The health bar displaying decreased health

Now let us see how we can programmatically show the number of available bullets in a player character's gun using a HUD. The following code sample adds a bullet and a bullet container texture to the game and programs their functionality. The code assumes that a project named "`XNABulletContainer`" is already created. The code uses the content member to load the bullet texture and `bulletContainer` texture from the XNA Framework Content Pipeline. This example assumes that the textures bullet and `bulletContainer` are available in the content project.

```
namespace XNABulletContainer
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
      /* texture for bullet image */
        Texture2D bulletImage;
      /* texture for bullet container */
        Texture2D bulletContainer;
        int MaxBullet = 24;
      /* current bullet count */
        int totalBullets;
      /* a variable to hold space between two consecutive
bullet images */
        float spaceBetweenImages = 0.0f;

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

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

             bulletImage = Content.Load<Texture2D>(@"bullet");
             bulletContainer = Content.Load<Texture2D>
             (@"bulletContainer");

             int maxBulletCanContain = bulletContainer.Height /
bulletImage.Height;

              /* calculate space between two bullet images */
              spaceBetweenImages = (float)maxBulletCanContain /
(float)totalBullets;
         }

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

              KeyboardState keyState = Keyboard.GetState();

              //If the Up Arrow is pressed, increase the number
of bullets
              if (keyState.IsKeyDown(Keys.Up) == true)
              {
                 totalBullets += 1;
              }

              //If the Down Arrow is pressed, decrease the number
of  bullets
              if (keyState.IsKeyDown(Keys.Down) == true)
              {
         totalBullets -= 1;
              }

         //Clamp the bullets between 0 and MaxBullet
              totalBullets = (int)MathHelper.Clamp(totalBullets,
0, MaxBullet);

              base.Update(gameTime);
         }

         protected override void Draw(GameTime gameTime)
         {

GraphicsDevice.Clear(Color.CornflowerBlue);

              spriteBatch.Begin();

              int postionY = 10;// this.Window.ClientBounds.
Height - bulletContainer.Height - 20;
              int positionX = 10;// this.Window.ClientBounds.
Width / 2 - bulletContainer.Width / 2;
```

```
//Draw the bullet container
                spriteBatch.Draw(bulletContainer, new
Rectangle(positionX,
                   postionY, bulletContainer.Width/2,
bulletContainer.Height),
                   new Rectangle(0, 0, bulletContainer.Width / 2,
                   bulletContainer.Height), Color.White);
                /* calculate first bullet position */
                Vector2 bulletPos = new Vector2(positionX + 2.0f,
bulletContainer.Height - bulletImage.Height );

//Draw Individual bullets
                for (int i = 0; i < totalBullets; i++)
                {
                   spriteBatch.Draw(bulletImage, bulletPos, Color.
White);
             /* Change the height of the next bullet in the
container */
                   bulletPos.Y -= (float)bulletImage.Height *
spaceBetweenImages;
                }

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

When you run the code, you notice that on pressing the up key, the bullet container shows an increase in the number of bullets (see Figure 6-3); and on pressing the down key, the bullet container shows a decrease in the number of bullets (see Figure 6-4).

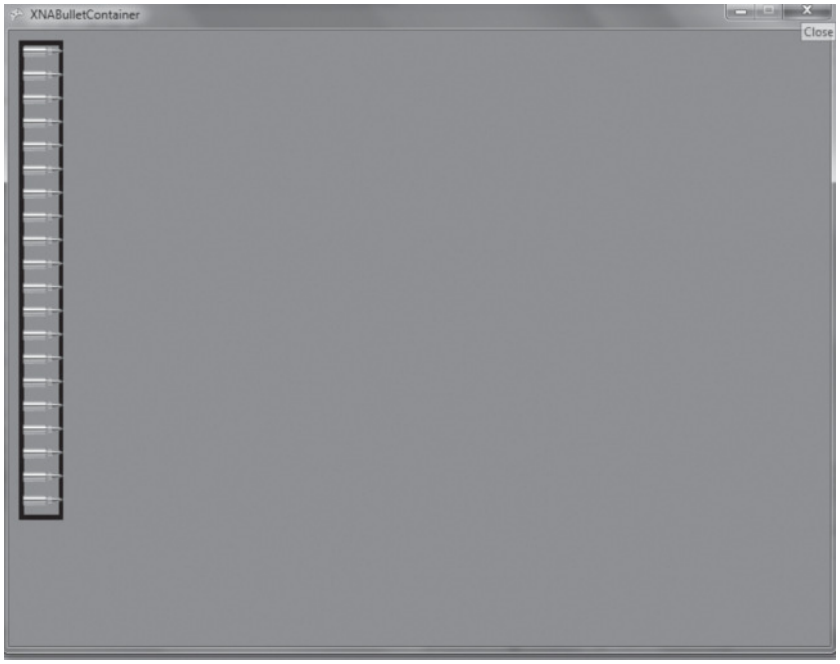


Figure 6-3

The bullet container with fully loaded bullets

Figure 6-4

The bullet container with a decreased number of bullets

Incorporating Artificial Intelligence (AI)

Incorporating AI involves programming the behavior of the nonplayer character (NPC) as close to a human as possible.

CERTIFICATION READY
What is the role of AI in games?
1.5

You already learned in Lesson 4, "Designing Specific Game Components," that AI designing in video games includes methods to produce the illusion of intelligence in the behavior of the NPCs. You also briefly learned about different strategies to incorporate AI in your games. In this lesson, we look further into different AI techniques and the algorithms for some of these techniques.

AI TECHNIQUES

You can add AI for various game scenarios. For example, you can use AI techniques to build an exciting "catch-me-if-you-can" plot or an element of surprise. Moreover, you can add life to your game characters by incorporating AI in their movements. These and many other objectives can be achieved through some clever use of common AI techniques and the programming logic behind these techniques. Some of the popular AI techniques are:

- Evading
- Chasing
- Flocking or grouping
- Path finding

Let us look at each of these techniques.

EVADING AI

One of the scenarios that presents an opportunity to incorporate AI is to make the movement of an object or NPC intelligent. For example, in a fantasy game, imagine a scenario in which a player character chases a token, such as a gold coin, whose movement is controlled by an NPC, such as an ogre. In this case, you can automate the coin sprite to disappear from one point to another

point on the screen as soon as the player character comes near it. This provides an illusion to the game player that the movement of the game object is controlled by a real human (monster) and not by the computer. Thus you can incorporate AI with the runaway sprite to make it seem realistic and challenging, yet defeatable. The following sample algorithm provides for an evading sprite.

1. Get the player character's position on the screen.
2. If the player character's position is in the defined range, then assign a random wait time to the sprite.
3. Do not move the sprite until the waiting time is over. This delay allows the player to catch the sprite if he reaches the sprite before the wait time is over.
4. As soon as the wait time is over, make the sprite appear in some other place on the scene.
5. Continue steps 2 to 4 until the player catches the sprite.

CHASING AI

Although an evading sprite has the player following the sprite, there might be a situation in a game in which the player is followed by a sprite and the player needs to evade the sprite. You require this technique when you have a computer-controlled or NPC enemy for the game player. You then program the enemy sprite to include the requisite AI so that the enemy character follows the player and behaves in response to the player just as a human enemy would. The following is a sample algorithm for incorporating chasing AI.

1. Assign a random waiting time to the AI before you plan its move.
2. Wait until the waiting time is over.
3. As soon as the waiting time is over, take the player position.
4. Calculate the difference between the player position and the AI position and decide on the AI direction based on the calculation.
5. Normalize the direction.
6. Add some randomness to the direction; otherwise, if the AI makes the same move as the player, it will be difficult for the player to win. You need to provide a balance by adding randomness to the AI.
7. Move towards the player according to the direction and with some speed.

FLOCKING OR GROUPING AI

Some games require specific AI techniques. You might feel that it is more realistic to let certain NPCs, such as a group of soldiers, move together. The AI logic behind this technique is the Craig Reynolds algorithm. This algorithm follows three simple rules:

- Steer and avoid colliding into other members of the group.
- Align towards the average heading of the members of the group.
- Steer or move toward the average position of the members of the group.

PATH FINDING AI

Apart from chasing and evading AI, another common AI in video games is the path finding AI. This AI involves moving a game object in an effort to find a path around an obstacle. Numerous algorithms are available for the path finding AI. The following topics briefly introduce you to two commonly available techniques:

- ***Breadcrumb path following:*** In this technique, the player character progresses through the game marking some invisible markers or "breadcrumbs" on her path unknowingly. You can program the NPC to look for and identify the breadcrumbs

and follow the path as laid by the player. As the player has already created the path, you do not need to worry about the complexity of the path and the number of obstacles in the path.

- ***Waypoint navigation:*** This technique is a good choice when your game has a large environment with multiple obstacles. Such an environment makes path finding a difficult and time-consuming task. Waypoint navigation allows you to place nodes in the game environment or, in other words, place reference points in the game world. You can then simply use these precalculated paths in the appropriate path finding algorithms.

Other techniques in path finding include terrain analysis, influence maps, and visibility graphs. ***Terrain analysis*** helps to increase the capabilities of an AI opponent by providing information about the game world. For example, information about hills, ambush points, water bodies, and so on can be used by the AI opponent to its advantage. An ***influence map*** is a technique that warns the computer-controlled opponent when its enemy is spotted. ***Visibility graphs*** break down larger areas into smaller areas that are interconnected. This technique is also used to give the game AI an advantage over the player.

Every game is unique and might require different AI techniques. Whatever the AI technique you follow, you must carefully build the AI into your game in a manner such that the player believes that he is playing against real opponents rather than computer-simulated characters. The AI should further motivate the player to continue playing the game. An AI opponent can easily become the curse of the game if you program it to be an extremely difficult opponent to defeat.

■ Handling Game Data

THE BOTTOM LINE

Handling game data involves capturing/retrieving the game data back and forth from a disk file and managing the state of the game. Capturing the game data helps in digitally storing the information about the progress of a game player in the respective game and retrieving the stored information at a later stage whenever the player wants to resume or reload the game from the saved point.

Game data means information about every element in a game. It includes the player or user data that encompasses the AI data and the level data. The AI data includes information about the position of the NPC and the position of the player character on the game screen. The level data includes details about the current game level. These details include the time elapsed since the start of the game level, the time setting (day/night) for the game scene, the weather of the game scene (cloudy/sunny), the settings chosen by the player, and so on.

Most video games today provide a mechanism to save the game data primarily for two reasons:

- It facilitates storing the player's progress in the current game session and reloading the game from the last saved point in the next session. This facility is a necessity because the complex and multilevel games of today take days to complete.
- Nonvolatile storage, achieved through devices such as cartridges, memory cards, and built-in hard drive consoles, has now become feasible. These storage devices store the game sessions at a low cost. To allure customers, game manufacturers today provide the player with the flexibility of saving the game at any point of time during the gameplay, at the end of each game level, or at the specific designated areas within the game.

Capturing User Data

CERTIFICATION READY
Why do you need to store and restore game data?
1.6

Capturing user data or game data is an important aspect of developing a game. This helps ensure that the score, the current level of the game, the characters or objects selected by the players, and any preference settings, such as sound and animation chosen by the player, are stored. As a result, when the players return to the game, they do not need to reset all the settings. They can also start from where they left the game. This helps the players move ahead in the game and keeps up their excitement level.

Imagine a situation in which you are playing a game that involves many levels that need to be completed to finish the game. When you start playing the game, you choose certain settings—for example, you can choose whether you want the music to play when the game is played, what the music should be, which character you want to represent, and what name you want to use for the character. You then go ahead with the game and finish the first level. This successful completion of Level 1 unlocks the next level and you start playing the next level. Midway into the second level, you quit the game to take care of some important errands. When you return to the game, you realize that all the settings that you made have been reset. Further, now you need to start from Level 1 and Level 2 is not yet unlocked. How will you feel? Of course you will be angry and frustrated, and you might not want to continue playing. To avoid such a situation, as a game developer, you need to have a provision for saving the game data when the players quit the game midway. Saving the game data helps ensure that when the players return to the game, they can start the game from where they left off. This helps keep the motivation levels of the players high and increases the chances of their playing the game till the end.

You can capture the game data by using the `XmlSerializer` and the `StorageContainer` classes in XNA 4.0. The `XmlSerializer` class serializes objects into XML documents and deserializes the objects from XML documents. Serializing an object means translating its public properties and fields into a serial format, such as an XML document for storage or transport. It is the way of saving an object's state into a stream or buffer. ***Deserialization*** is the process of restoring the object to its original state from the serialized form. The `StorageContainer` class is a logical set of storage files. You can create files to store the state of the various objects and components in the game and store the files in a `StorageContainer` object.

Storing the Game Data

CERTIFICATION READY
What is the mechanism to store game data using XNA 4.0?
1.7

To store the game data, you need to write code to perform the following tasks:

1. **Define the game data to be stored:** Before you save the game data, you need to decide on the type of game data you want to store. That is, you need to decide on the objects and components of the game that you want to store when the player saves the game. You can include as many objects and components as you want. For example, in a shooter game, you can store information such as level, score, the character name chosen by the player, the position of the player, the status of ammunition in hand, and the gun the player uses.

   After you decide the objects and components to be stored, create a class or a structure that defines the important objects and components you have identified.

2. **Serialize the game data into the required game file:** Now that you have defined the data you want to store, you need to create a ***save game file*** to store the data using the `StorageContainer` class. To ensure that only the latest game data is stored, you need to check whether a game file has already been created and saved earlier. If a save game

file already exists, you need to delete the existing file and then create a new file. To check whether a file with a specific name exists, use the `FileExists` method of the `StorageContainer` class, and to delete an existing file from the container, use the `DeleteFile` method of the `StorageContainer` class. Finally, serialize the data to be stored and send it as a stream into the save game file.

The following steps show the code required to save the game data. As already discussed, the `StorageContainer` class represents the logical collection of storage files. To get the physical storage device, such as a memory unit or hard drive, you need to use the `StorageDevice` class and its methods. The following code sample assumes that a `StorageDevice` object named "device" is obtained.

MORE INFORMATION

For more information on obtaining a storage device, refer to the "Reading and Writing Data Files" section in the MSDN Library.

SAVE DATA TO A SAVE GAME FILE

GET READY. Store the game data into a save game file.

1. Define the game data that you want to save. For this, you define a new class or structure.

```
public struct PlayerData
{
    public string PlayerName;
    public Vector2PlayerPosition;
    public int Level;
    public int Score;
    public List<string> completedAchievements;
/* If game is having 24 hr day night system we need to save
that.*/
    public System.TimeSpan currentGameTime;
/* If your game has some weather condition , need to save it too
. My game has a fog effect.*/
    public bool fogEnable;
}
```

2. Serialize the data in the save data file. To achieve this, perform the following steps:

   a. Create a `StorageContainer` object to access the specified device:

```
// Open a storage container.

IAsyncResult asyncResult=
   device.BeginOpenContainer("SavingPlayerProfile",
null, null);

// Wait for the WaitHandle to become signaled.
asyncResult.AsyncWaitHandle.WaitOne();

StorageContainer container = device.EndOpenContainer
(asyncResult);

// Close the wait handle.
asyncResult.AsyncWaitHandle.Close();
```

   b. Call `FileExists` to determine whether the save game file already exists. If it exists, then call the `DeleteFile` method to delete it. The following code performs this:

```
string filename = "savedGameState.sav";
```

```
// Check to see whether the save exists.
if (container.FileExists(filename))
    // Delete it so that we can create one fresh.
    container.DeleteFile(filename);
```

c. Create a `Stream` object on the file by using the `CreateFile` method, as shown in the following code:

```
// Create the file.
Stream fileStream = container.CreateFile(filename);
```

d. Create an `XmlSerializer` object and pass the type of the structure that defines your save game data, as shown in the following code.

```
// Convert the object to XML data and put it in the stream.
XmlSerializer serializer = new
XmlSerializer(typeof(PlayerData));
```

e. Call `Serialize`, and then pass the `Stream` and the data to serialize. The `XmlSerializer` converts data in the structure to XML and uses the `Stream` to write the data into the file.

```
PlayerData playerData = new PlayerData();

/* Then set playerData with appropriate info from game */

playerData.fogEnable = true;

playerData. PlayerName = "your name";

playerData.Score = 300;

serializer.Serialize(fileStream, playerdata);
```

f. Close the `Stream` and dispose the `StorageContainer` object. This commits the changes made to the device.

```
// Close the file.
fileStream.Close();
//dispose the containder
Container.Dispose();
```

Now that you have saved the game data, let us understand how to read the saved game data.

Loading the Game Data

CERTIFICATION READY
What is the procedure to restore the saved user data in XNA 4.0?
1.7

Loading or reading the game data from the game file involves the following tasks:

1. **Create a `StorageContainer` object to access the game file:** To start loading the save game state, you need to create an object of the `StorageContainer` class to access the device and open the game file that contains the stored game data. You also need to check whether any game files exist. If the player is playing the game for the first time, a game file will not be present. In case a game file does not exist, you must dispose the container.
2. **Deserialize the game data to load:** You have created a `StorageContainer` object and checked whether a game file exists. If a game file exists, you need to load the game data so that the players can start from where they left the game previously. To do this, you first create a stream and then create an `XmlSerializer` object that will access the game data stored in the file through the stream. You then call the `Deserialize` method of the `XmSerializer` object. This method accesses the game data and converts the data back from XML format to the actual format as required in the game. Finally, you close the stream and dispose of the `StorageContainer` object.

Let us look at how to write code to perform each of these tasks.

READ DATA FROM A SAVE GAME FILE

GET READY. Restore the game data from the save game file.

1. Create a `StorageContainer` object to access the specified device.

```
// Open a storage container.
IAsyncResult asyncResult=
    device.BeginOpenContainer("SavingPlayerProfile",
null, null);

// Wait for the WaitHandle to become signaled.
asyncResult.AsyncWaitHandle.WaitOne();

StorageContainer container =
device.EndOpenContainer(asyncResult);

// Close the wait handle.asyncResult.AsyncWaitHandle.Close();
```

2. Call `FileExists` to determine whether the saved game file exists.

```
string filename = "savedGameState.sav";

// Check to see whether the save exists.
if (!container.FileExists(filename))
{
    // If not, dispose of the container and return.
    container.Dispose();
    return;
}
```

3. Open a `Stream` object on the file by using the `OpenFile` method, as shown in the following code.

```
// Open the file.
Stream fileStream = container.OpenFile(filename,
   FileMode.Open);
```

4. Create an `XmlSerializer` object and pass the type of the structure that defines your save game data.

```
XmlSerializer serializer = new
XmlSerializer(typeof(PlayerData));
```

5. Call the `Deserialize` method and pass the `Stream` object. The `Deserialize` method returns a copy of the save game structure populated with the data from the save game file. Note that you need to cast the return value from `Object` to the respective type, as shown in the following code. (In this case, the type is `PlayerData`.)

```
PlayerData data = (PlayerData)serializer.
   Deserialize(fileStream);
```

6. Close the `Stream` and dispose the storage container.

```
// Close the file.
fileStream.Close();
// Dispose of the container.

container.Dispose();
```

Saving and restoring game data tracks the player's progress in a game. However, to track the game flow while your game is running, you need to define the possible states for your game and manage these states. The following section discusses how to manage the game states that you define for your game.

Managing Game States

Game state is a logical concept that helps you to track the flow of the game.

CERTIFICATION READY
How will you manage the flow of a fantasy game using game states?
1.6

If you recall, a game state defines the behavior of every object at any given point in the game. In general, the simplest of games can have the following game states:

- **Initialize:** In this state, all the necessary resources, such as the required services or any nongraphics resources, are loaded into the memory.
- **DisplayMenu:** In this state, the main menu is displayed where the player can view the demo game or start the game play.
- `PlayerPlay`: In this state, the player plays the game and in the course of gameplay, any one of three things can happen:
  - The player is left with more chances and, as a result, the game once again enters the Player Play State.
  - The player is void of chances and, as a result, the game state enters the Game Over State.
  - The player loses the game and the game enters the Player Lose State.
- `PlayerLose`: In this state, the player can choose to replay the game.
- `GameOver`: In this state, the player can choose to replay the game.

You can see that the game tends to behave according to the value of the game state. You manage these game states according to the player action. For example, consider the game states that we discussed at the start of this topic. Based on the player action, the game enters into the different states. Based on these states, you program your game to progress forward or backward. For example, when your game starts, the state is automatically set to the `Initialize` state. Once the initialization completes, the game progresses to the next state, which is the `DisplayMenu` state. In this state, depending on the action performed by the player, your game enters the `PlayerPlay` state. Likewise, as and when the player progresses in the game, you should program to set the game state variable to the current game state. Additionally, you need to track the game flow according to the current value of the game state variable. Suppose the current state of the state variable is the `PlayerPlay` state, and if in this state, the player loses the game, then you need to set the game state variable to the `PlayerLose` state and this state becomes the current game state. In this state, you can define the behavior of the game. That is, the game can display different options, such as an option to navigate the player to the opening screen or an option to start the game again.

The following code sample defines the possible states of a game. It then manages the flow of the game, based on the current state of the game. The code assumes that a project named "XNAGameState" is created already.

```
namespace XNAGameState
{
    public enum GameState
    {
        Initialize,
        DisplayMenu,
        PlayerPlay,
        PlayerLose,
```

```
            GameOver,
        };

    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        /* Assign default game state value to GameState.
Initialize */

        GameState currentGameState = GameState.Initialize;

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

        }


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

        switch (currentGameState)
        {
            case GameState.Initialize:
                InitializeScene();
                break;
            case GameState.DisplayMenu:
                showMenu();
                break;
            case GameState.PlayerPlay:
                updatePlayerGamePlayLoop(gameTime);
                break;
            case GameState.PlayerLose:
                if (player.IsReplay)
                {
                     player.Reinitialize();
                     currentGameState = GameState.PlayerPlay;
                }
                else
                {

                     currentGameState = GameState.GameOver;
                }
            case GameState.GameOver:
                if (player.IsReplay)
                {
                     player.Reinitialize();
                     currentGameState = GameState.PlayerPlay;
                }
                else
```

```
                {
                    UnloadContent();
                    currentGameState = GameState.DisplayMenu;
                }
            }
            base.Update(gameTime);
        }
        public void InitializeScene()
        {
            /* Initialize your scene here and when scene
initialization finishes set state to DisplayMenu */
                    currentGameState = GameState.DisplayMenu;
        }
        public void showMenu()
        {
            //Enable rendering of Menu
        }
        public void updatePlayerGamePlayLoop(GameTime gameTime)
        {
            //Update Player Game play
        }
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            base.Draw(gameTime);
        }
    }
}
```

SKILL SUMMARY

In This Lesson, You Learned:

- The game development process also involves creating tools to handle game specific tasks.
- Tools, such as an asset converter, level editor, and map editor, assist the game creation process.
- You can define the behavior of the components added to your game by adding the necessary game code.
- Some of the popular AI techniques are chasing AI, evading AI, flocking or grouping AI, and path finding AI.
- Flocking or grouping AI involves the Craig Reynolds algorithm.
- Popular path finding techniques include breadcrumb path following and waypoint navigation.
- Handling game data means capturing and retrieving the game data to and from a disk file. It also involves managing the state of the game.
- Game data means the information about every element in a game, such as the AI data and the level data. AI data pertains to the position of the AI and the player character. Level data includes information about the particular level.

- Saving a game helps the player to prevent the loss of progress in the game when he needs to quit the game due to an interruption or for taking a break when the game session is long.
- You can capture and retrieve the game data by using the `XmlSerializer` and the `StorageContainer` classes in XNA 4.0.
- Defining game states helps you track the flow in your game.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ assist the development of video games.
2. _____ AI technique involves the Craig Reynolds algorithm.
3. _____ and _____ are the tools that might be released with the final game.
4. The _____ helps to convert the artwork into formats required by the game.
5. _____ method helps to load a texture from the XNA Framework Content Pipeline.
6. _____ class helps you retrieve the physical storage device to save the game data.
7. _____ class helps you serialize and deserialize objects to and from XML document.
8. _____ is a logical concept that helps you track the flow of a game.
9. You can provide an illusion of intelligence in nonplayer characters by incorporating _____ techniques.
10. _____ and _____ are popular path finding AI techniques.

Multiple Choice

1. Which of the following objects and methods will you use when saving a save game file? (Choose all that apply.)
   a. `StorageContainer` object
   b. `Stream` object
   c. `XmlSerializer.Serialize` method
   d. `XmlSerializer.Deserialize` method
2. Which of the following AI techniques places nodes in the game world and uses them in the path finding algorithms?
   a. Waypoint navigation
   b. Breadcrumb path following
   c. Evading AI
   d. Terrain analysis
3. Which of the following classes helps you get a storage device to store the game data?
   a. `StorageDevice`
   b. `StorageContainer`
   c. `XmlSerializer`
   d. `Stream`

4. Which of the following should you perform with a StorageContainer object to commit the changes made to the respective storage device?
   a. Call the StorageContainer.Dispose method
   b. Close the Stream object.
   c. Call the XmlSerialize.Serialize method
   d. Call the StorageContainer.CreateFile method

5. Which of the following classes helps you draw sprites on the game screen?
   a. Texture 2D
   b. SpriteBatch
   c. GraphicsDeviceManager
   d. ContentManager

6. Which of the following classes holds sprites?
   a. Texture 2D
   b. SpriteBatch
   c. GraphicsDeviceManager
   d. ContentManager

7. Which of the following options helps you to track the player's progress in a game?
   a. Saving game data
   b. Maintaining game state
   c. Adding game objects to content pipeline
   d. Creating a level editor tool

8. Which of the following game tools helps the game player to customize or modify levels within a game?
   a. Level editor
   b. Map editor
   c. Asset conversion tool
   d. 3D Studio Max

9. Which of the following activities do you have to perform while loading the game data from a save game file? (Choose all that apply.)
   a. Serialize data
   b. Deserialize data
   c. Create a storage container object to access the game file
   d. Define the game data to be stored

10. You develop a role-playing game. You decide to include a level editor along with the release of the game. Which of the following activities involves the creation of the level editor for your game?
    a. Tool creation
    b. Programming game components
    c. Game Designing
    d. Incorporating artificial intelligence

■ Competency Assessment

Scenario 6-1: Creating Tools

You are creating a racing game. You decide to provide your game player with the option of manipulating the racing track. How would you achieve this?

Scenario 6-2: Incorporating AI

You are creating a fantasy game. Your game involves a scenario in which the player character needs to escape from the attacks of an NPC. You decide to provide a realistic challenge to the player by making your NPC behave in response to the player. How would you achieve this?

■ Proficiency Assessment

Scenario 6-3: Programming Components

You are creating a first-person shooter (FPS) game. For a particular scene, your game should inform the player of the number of tools in his inventory.

1. How would you recommend this problem be solved?
2. How would you programmatically solve this?

Scenario 6-4: Implementing Game Saves

You are developing a fantasy game. You decide to track the game player's progress at the end of each level.

1. How would you recommend this problem be solved?
2. How would you programmatically solve this?

Appendix A
Gaming Development Fundamentals: Exam 98-374

Objective Domain	Skill Number	Lesson Number
Understanding Game Design		
Differentiate among game types.	1.1	1
Differentiate among game genres.	1.2	1
Understand player motivation.	1.3	1
Design the user interface.	1.4	3, 5
Understand components.	1.5	6
Capture user data.	1.6	5, 6
Work with XNA.	1.7	5, 6
Understand Hardware		
Choose an input device.	2.1	2
Choose an output device.	2.2	2
Work with the network.	2.3	2
Manage game performance.	2.4	2
Understand the different game platforms.	2.5	1, 2
Understand Graphics		
Understand rendering engines.	3.1	3
Plan for game state.	3.2	4, 5
Draw objects.	3.3	3
Understand Animation		
Animate basic characters.	4.1	4
Transform objects.	4.2	4
Work with collisions.	4.3	4

Index

Entries in *italics* refer to game names

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

X

Z