Windows Phone Mango编程实践

***Windows Phone Mango Programming Practice***

第三篇 XNA游戏篇

# XNA二维动作游戏开发

《礼记·中庸》：“凡事豫则立，不豫则废。言前定，则不跲；事前定，则不困；行前定，则不疚；道前定，则不穷。”任何事情，有准备就能成功，没有任何准备就会失败。话在事先准备好了，辩论时就不会理屈词穷了；在办事前先做好充分的准备，到时候就不至于处于困境了；在行动前先做好了充分的准备，到时候就不会感到内疚了；履行做人的原则，有准备了就不至于有什么不顺畅之事了。游戏的设计同样如此，如果游戏开发的中途修改游戏的设计初衷或者理念，将会付出昂贵的代价，甚至决定项目最终的成败。孙子曰：“夫未战而庙算胜者，得算多也；未战而庙算不胜者，得算少也。多算胜，少算不胜，而况于无算乎！”讲述的也是这个道理。

## 游戏设计之初的思考

本章参考和引用APP HUB（<http://create.msdn.com/en-US>）的Game Development Tutorial，以及MSDN Windows Phone开发文档。

在游戏应用程序编码开始之前，首先问自己几个关于游戏设计问题，想清楚确定好目标后再着手开始设计开发。

* 它使什么类型的游戏？
* 游戏的目标是什么？
* 游戏的玩法是设计？
* 游戏的采用何种驱动？
* 游戏的艺术资源如何设计？

《礼记·中庸》：“凡事豫则立，不豫则废。言前定，则不跲；事前定，则不困；行前定，则不疚；道前定，则不穷。”任何事情，有准备就能成功，没有任何准备就会失败。话在事先准备好了，辩论时就不会理屈词穷了；在办事前先做好充分的准备，到时候就不至于处于困境了；在行动前先做好了充分的准备，到时候就不会感到内疚了；履行做人的原则，有准备了就不至于有什么不顺畅之事了。游戏的设计同样如此，如果游戏开发的中途修改游戏的设计初衷或者理念，将会付出昂贵的代价。孙子曰：“夫未战而庙算胜者，得算多也；未战而庙算不胜者，得算少也。多算胜，少算不胜，而况于无算乎！”讲述的也是这个道理。

### 游戏设计流程图

创建游戏设计文档有助于减轻潜在的缺陷，帮助开发团队中的成员理解和处理的游戏的设计逻辑。理解我们要设计的游戏逻辑，就从下面的流程图中开始。



图16-1 流程图

您可以看出，小小的射击游戏的流程图就如此繁杂，大型游戏的设计则更为复杂。详细的文档作为沟通工具可以帮助设计人员、开发人员和测试人员理解期望的目标，完成预期的工作。

## 创建游戏角色

### 新建游戏应用程序

确认已经安装Windows Phone Mango开发平台，详细信息请见[Installing Windows Phone Developer Tools](http://msdn.microsoft.com/en-us/library/ff402530(v=VS.92).aspx)。启动Visual Studio 2010 Express或者Visual Studio 2010。如果出现注册窗体，您可以注册或暂时关闭该窗体。

通过菜单创建新应用程序，选择**File | New Project**。在弹出的窗体中选择**[XNA Game Studio 4.0] | [Windows Phone Game(4.0)]**，在项目名称栏中填入**Shooter**。

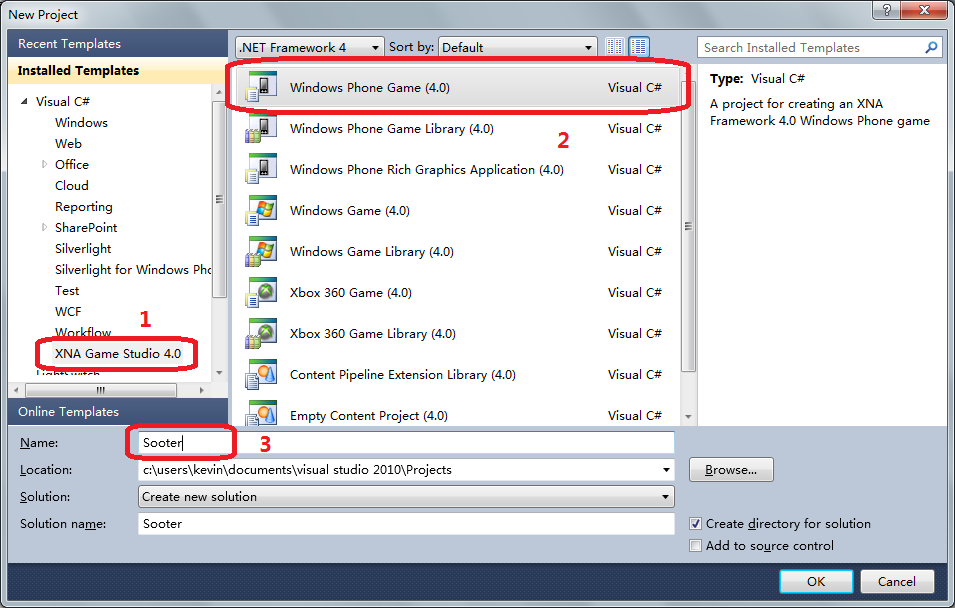


图16-2 新建项目

### 游戏角色—飞艇

创建游戏角色类飞艇，类名为Player。同时按下SHIFT + ALT + C，在弹出的窗体中选择**[Code]|[Code file]**。

在新创建的Player.cs文件中添加引用。

XNA Project: Shooter File: Player.cs

using System;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

在Player类中添加游戏用户的变量，存储用户数据。包括动画、位置二维坐标、状态、游戏用户生命值。这些数据组合在一起绘制游戏角色，在给定的二维向量Vector2 来绘制二维图形Texture2D，并使用Health变量记录游戏角色的生命值。

XNA Project: Shooter File: Player.cs

// Animation representing the player

public Animation PlayerAnimation;

// Position of the Player relative to the upper left side of the screen

public Vector2 Position;

// State of the player

public bool Active;

// Amount of hit points that player has

public int Health;

// Get the width of the player ship

public int Width

{

get { return PlayerAnimation.FrameWidth; }

}

// Get the height of the player ship

public int Height

{

get { return PlayerAnimation.FrameHeight; }

}

Initialize方法初始化游戏角色；Update方法更新游戏角色的位置；Draw方法调用SpriteBatch.Draw方法渲染画面。

XNA Project: Shooter File: Player.cs

// Initialize the player

public void Initialize(Animation animation, Vector2 position)

{

PlayerAnimation = animation;

// Set the starting position of the player around the middle of the screen and to the back

Position = position;

// Set the player to be active

Active = true;

// Set the player health

Health = 100;

}

// Update the player animation

public void Update(GameTime gameTime)

{

PlayerAnimation.Position = Position;

PlayerAnimation.Update(gameTime);

}

// Draw the player

public void Draw(SpriteBatch spriteBatch)

{

PlayerAnimation.Draw(spriteBatch);

}

我们使用简单的动画实现游戏角色的创建，现在我们就开始编写游戏角色动画类Animation类的编码。同时按下SHIFT + ALT + C，在弹出的窗体中选择**[Code]|[Code file]**。

在新创建的Animation.cs文件中添加引用。

XNA Project: Shooter File: Animation.cs

// Animation.cs

//Using declarations

using System;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Content;

using Microsoft.Xna.Framework.Graphics;

添加控制游戏角色的变量。

XNA Project: Shooter File: Animation.cs

// The image representing the collection of images used for animation

Texture2D spriteStrip;

// The scale used to display the sprite strip

float scale;

// The time since we last updated the frame

int elapsedTime;

// The time we display a frame until the next one

int frameTime;

// The number of frames that the animation contains

int frameCount;

// The index of the current frame we are displaying

int currentFrame;

// The color of the frame we will be displaying

Color color;

// The area of the image strip we want to display

Rectangle sourceRect = new Rectangle();

// The area where we want to display the image strip in the game

Rectangle destinationRect = new Rectangle();

// Width of a given frame

public int FrameWidth;

// Height of a given frame

public int FrameHeight;

// The state of the Animation

public bool Active;

// Determines if the animation will keep playing or deactivate after one run

public bool Looping;

// Width of a given frame

public Vector2 Position;

Initialize方法通过传递的参数初始化。

XNA Project: Shooter File: Animation.cs

public void Initialize(Texture2D texture, Vector2 position,

int frameWidth, int frameHeight, int frameCount,

int frametime, Color color, float scale, bool looping)

{

// Keep a local copy of the values passed in

this.color = color;

this.FrameWidth = frameWidth;

this.FrameHeight = frameHeight;

this.frameCount = frameCount;

this.frameTime = frametime;

this.scale = scale;

Looping = looping;

Position = position;

spriteStrip = texture;

// Set the time to zero

elapsedTime = 0;

currentFrame = 0;

// Set the Animation to active by default

Active = true;

}

Update方法对GameTime进行操作，计算实际像素的帧数。

XNA Project: Shooter File: Animation.cs

public void Update(GameTime gameTime)

{

// Do not update the game if we are not active

if (Active == false)

return;

// Update the elapsed time

elapsedTime += (int)gameTime.ElapsedGameTime.TotalMilliseconds;

// If the elapsed time is larger than the frame time

// we need to switch frames

if (elapsedTime > frameTime)

{

// Move to the next frame

currentFrame++;

// If the currentFrame is equal to frameCount reset currentFrame to zero

if (currentFrame == frameCount)

{

currentFrame = 0;

// If we are not looping deactivate the animation

if (Looping == false)

Active = false;

}

// Reset the elapsed time to zero

elapsedTime = 0;

}

// Grab the correct frame in the image strip by multiplying the currentFrame index by the frame width

sourceRect = new Rectangle(currentFrame \* FrameWidth, 0, FrameWidth, FrameHeight);

// Grab the correct frame in the image strip by multiplying the currentFrame index by the frame width

destinationRect = new Rectangle((int)Position.X - (int)(FrameWidth \* scale) / 2,

(int)Position.Y - (int)(FrameHeight \* scale) / 2,

(int)(FrameWidth \* scale),

(int)(FrameHeight \* scale));

}

Draw方法中调用spriteBatch.Draw渲染画面。

XNA Project: Shooter File: Animation.cs

// Draw the Animation Strip

public void Draw(SpriteBatch spriteBatch)

{

// Only draw the animation when we are active

if (Active)

{

spriteBatch.Draw(spriteStrip, destinationRect, sourceRect, color);

}

}

在Game1.cs的LoadContent方法中加载游戏角色。

XNA Project: Shooter File: Game1.cs

// Create a new SpriteBatch, which can be used to draw textures.

spriteBatch = new SpriteBatch(GraphicsDevice);

// Load the player resources

Animation playerAnimation = new Animation();

Texture2D playerTexture = Content.Load<Texture2D>("shipAnimation");

playerAnimation.Initialize(playerTexture, Vector2.Zero, 115, 69, 8, 30, Color.White, 1f, true);

Vector2 playerPosition = new Vector2(GraphicsDevice.Viewport.TitleSafeArea.X, GraphicsDevice.Viewport.TitleSafeArea.Y

+ GraphicsDevice.Viewport.TitleSafeArea.Height / 2);

player.Initialize(playerAnimation, playerPosition);

为传递游戏循环的更新时间，需要在Game1.cs的Update方法中调用player.Update方法，将游戏循环的更新时间通过Player类传递给Animation类。

XNA Project: Shooter File: Game1.cs

private void UpdatePlayer(GameTime gameTime)

{

player.Update(gameTime);

……

}

### 飞艇的控制

XNA支持的游戏平台包括Windows、XBOX 360和Windows Phone。Windows平台的输入设备为键盘和鼠标；XBOX 360的输入设备为游戏手柄或者其他控制器；Windows Phone的输入采用触控感应和重力传感器。我们将采用相同的触控输入代码来支持三种平台。

在Game1.cs中添加Windows Phone手指触控的引用。

XNA Project: Shooter File: Game1.cs

using Microsoft.Xna.Framework.Input.Touch;

在Game1的类中添加对于键盘和游戏手柄的输入的支持，以及移动速度的变量。

XNA Project: Shooter File: Game1.cs

// Keyboard states used to determine key presses

KeyboardState currentKeyboardState;

KeyboardState previousKeyboardState;

// Gamepad states used to determine button presses

GamePadState currentGamePadState;

GamePadState previousGamePadState;

// A movement speed for the player

float playerMoveSpeed;

在Game1类的Initialize方法中设置游戏角色移动的速度，设置触控面板的手势识别为**FreeDrag**。TouchPanel.EnabledGestures是枚举类型，在此设置触控系统只关注FreeDrag的手势。

XNA Project: Shooter File: Game1.cs

// Set a constant player move speed

playerMoveSpeed = 8.0f;

//Enable the FreeDrag gesture.

TouchPanel.EnabledGestures = GestureType.FreeDrag;

创建UpdatePlayer方法，封装触控输入的识别，实现控制游戏角色的在Windows、XBOX 360和Windows Phone三种平台运行时的移动控制。

在键盘上，它会检查是否向上，向下、 向左或向右箭头键被按下，如果是则更新游戏玩家角色的位置，且固定的速度移动。

在游戏手柄中，以thumbstick与速度的乘积为更新后位置的计算值。使用thumbsticks，玩家可以选择如何快速移动。

在Windows Phone平台中，检查手指触摸输入的位置差来计算更新后的游戏角色的位置，通过手指触控，游戏角色移动的速度取决于手指自由拖动的速度。

XNA Project: Shooter File: Game1.cs

private void UpdatePlayer(GameTime gameTime)

{

player.Update(gameTime);

// Windows Phone Controls

while (TouchPanel.IsGestureAvailable)

{

GestureSample gesture = TouchPanel.ReadGesture();

if (gesture.GestureType == GestureType.FreeDrag)

{

player.Position += gesture.Delta;

}

}

// Get Thumbstick Controls

player.Position.X += currentGamePadState.ThumbSticks.Left.X \* playerMoveSpeed;

player.Position.Y -= currentGamePadState.ThumbSticks.Left.Y \* playerMoveSpeed;

// Use the Keyboard / Dpad

if (currentKeyboardState.IsKeyDown(Keys.Left) ||

currentGamePadState.DPad.Left == ButtonState.Pressed)

{

player.Position.X -= playerMoveSpeed;

}

if (currentKeyboardState.IsKeyDown(Keys.Right) ||

currentGamePadState.DPad.Right == ButtonState.Pressed)

{

player.Position.X += playerMoveSpeed;

}

if (currentKeyboardState.IsKeyDown(Keys.Up) ||

currentGamePadState.DPad.Up == ButtonState.Pressed)

{

player.Position.Y -= playerMoveSpeed;

}

if (currentKeyboardState.IsKeyDown(Keys.Down) ||

currentGamePadState.DPad.Down == ButtonState.Pressed)

{

player.Position.Y += playerMoveSpeed;

}

// Make sure that the player does not go out of bounds

player.Position.X = MathHelper.Clamp(player.Position.X, 0, GraphicsDevice.Viewport.Width - player.Width);

player.Position.Y = MathHelper.Clamp(player.Position.Y, 0, GraphicsDevice.Viewport.Height - player.Height);

}

每次游戏循环中以相同的方式更新每一帧，调用Game1的**Update()**方法，在Update调用**UpdatePlayer()**方法。

XNA Project: Shooter File: Game1.cs

/// <summary>

/// Allows the game to run logic such as updating the world,

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Update(GameTime gameTime)

{

// Allows the game to exit

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

this.Exit();

// Save the previous state of the keyboard and game pad so we can determinesingle key/button presses

previousGamePadState = currentGamePadState;

previousKeyboardState = currentKeyboardState;

// Read the current state of the keyboard and gamepad and store it

currentKeyboardState = Keyboard.GetState();

currentGamePadState = GamePad.GetState(PlayerIndex.One);

//Update the player

UpdatePlayer(gameTime);

base.Update(gameTime);

}

### 游戏的视差背景

飞艇在移动时可以绘制云图案作为背景，并将其从右向左移动。本例中我们采用视差背景实现真实动感的背景效果。

天文学中采用视差法是确定天体之间距离。视差就是从有一定距离的两个点上观察同一个目标所产生的方向差异。从目标看两个点之间的夹角，叫做这两个点的视差角，两点之间的距离称作基线。只要知道视差角度和基线长度，就可以计算出目标和观测者之间的距离。

游戏开发中利用视觉上的误差，即通常所说的视错觉。造成所谓视差的不仅是人们肉眼所存在的局限和障碍，同时也是由于文化深层的联想和思考所引发。本例中绘制视差背景的方法是：用于绘制多层图像并以不同的速度移动，来达到在飞艇在云端飞行的视错觉。

按 SHIFT + ALT + C创建视差背景类，键入类名ParallaxingBackground.cs。

创建游戏角色类，类名为ParallaxingBackground。同时按下SHIFT + ALT + C，在弹出的窗体中选择**[Code]|[Code file]**。

在新创建的ParallaxingBackground.cs文件中添加引用。

XNA Project: Shooter File: ParallaxingBackground.cs

using System;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Content;

using Microsoft.Xna.Framework.Graphics;

在ParallaxingBackground类中定义类型为Texture2D的背景图片变量、Vector2数组和移动速度的变量。

XNA Project: Shooter File: ParallaxingBackground.cs

// The image representing the parallaxing background

Texture2D texture;

// An array of positions of the parallaxing background

Vector2[] positions;

// The speed which the background is moving

int speed;

在Initialize()方法中，加载背景图片，设定图片移动速度。

首先使用content.load方法初始化图形，然后计算Vector2数组中对象的数量(screenWidth / texture.Width + 1)，其中+1的目的是为保证背景切换平滑。在For循环中，设定背景图片显示的初始位置。

XNA Project: Shooter File: ParallaxingBackground.cs

public void Initialize(ContentManager content, String texturePath, int screenWidth, int speed)

{

// Load the background texture we will be using

texture = content.Load<Texture2D>(texturePath);

// Set the speed of the background

this.speed = speed;

// If we divide the screen with the texture width then we can determine the number of tiles need.

// We add 1 to it so that we won't have a gap in the tiling

positions = new Vector2[screenWidth / texture.Width + 1];

// Set the initial positions of the parallaxing background

for (int i = 0; i < positions.Length; i++)

{

// We need the tiles to be side by side to create a tiling effect

positions[i] = new Vector2(i \* texture.Width, 0);

}

}

在Update方法中改变背景图片的位置坐标。每个背景图片被认为是一个瓷片Tile，更新瓷片的X轴坐标，以移动速度变量值作为X轴坐标移动的增量。如果移动速度变量小于零，则瓷片从右往左移动。如果移动速度变量大于零，则瓷片从左往右移动。移动时判断瓷片显示位置是否超出屏幕区域，如果是则重置瓷片的X轴坐标，以便背景滚动平滑。

XNA Project: Shooter File: ParallaxingBackground.cs

public void Update()

{

// Update the positions of the background

for (int i = 0; i < positions.Length; i++)

{

// Update the position of the screen by adding the speed

positions[i].X += speed;

// If the speed has the background moving to the left

if (speed <= 0)

{

// Check the texture is out of view then put that texture at the end of the screen

if (positions[i].X <= -texture.Width)

{

positions[i].X = texture.Width \* (positions.Length - 1);

}

}

// If the speed has the background moving to the right

else

{

// Check if the texture is out of view then position it to the start of the screen

if (positions[i].X >= texture.Width \* (positions.Length - 1))

{

positions[i].X = -texture.Width;

}

}

}

}

位置坐标更新完毕后，使用Draw方法绘制视差背景。

XNA Project: Shooter File: ParallaxingBackground.cs

public void Draw(SpriteBatch spriteBatch)

{

for (int i = 0; i < positions.Length; i++)

{

spriteBatch.Draw(texture, positions[i], Color.White);

}

}

在Game1.cs中声明视差背景的层bgLayer1和bgLayer2。

XNA Project: Shooter File: Game1.cs

// Image used to display the static background

Texture2D mainBackground;

// Parallaxing Layers

ParallaxingBackground bgLayer1;

ParallaxingBackground bgLayer2;

在Game1的Initialize方法中实例化视差背景层。

XNA Project: Shooter File: Game1.cs

bgLayer1 = new ParallaxingBackground();

bgLayer2 = new ParallaxingBackground();

在Game1的LoadContent方法中加载背景层。

XNA Project: Shooter File: Game1.cs

// Load the parallaxing background

bgLayer1.Initialize(Content, "bgLayer1", GraphicsDevice.Viewport.Width, -1);

bgLayer2.Initialize(Content, "bgLayer2", GraphicsDevice.Viewport.Width, -2);

mainBackground = Content.Load<Texture2D>("mainbackground");

在Game1的Update方法中，更新游戏角色飞艇之后更新视差背景。

XNA Project: Shooter File: Game1.cs

// Update the parallaxing background

bgLayer1.Update();

bgLayer2.Update();

在Game1的Draw方法中，绘制游戏背景。

XNA Project: Shooter File: Game1.cs

GraphicsDevice.Clear(Color.CornflowerBlue);

// Start drawing

spriteBatch.Begin();

spriteBatch.Draw(mainBackground, Vector2.Zero, Color.White);

// Draw the moving background

bgLayer1.Draw(spriteBatch);

bgLayer2.Draw(spriteBatch);

### 创建万恶的敌人

为了让飞艇英雄有用武之地，我们设置飞艇英雄拯救地球的障碍—创建万恶的敌人。

创建游戏角色类飞艇，类名为Enemy。同时按下SHIFT + ALT + C，在弹出的窗体中选择**[Code]|[Code file]**。

在新创建的Enemy.cs文件中添加引用。

XNA Project: Shooter File: Enemy.cs

using System;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

添加敌人的控制变量。

XNA Project: Shooter File: Enemy.cs

// Animation representing the enemy

public Animation EnemyAnimation;

// The position of the enemy ship relative to the top left corner of thescreen

public Vector2 Position;

// The state of the Enemy Ship

public bool Active;

// The hit points of the enemy, if this goes to zero the enemy dies

public int Health;

// The amount of damage the enemy inflicts on the player ship

public int Damage;

// The amount of score the enemy will give to the player

public int Value;

// Get the width of the enemy ship

public int Width

{

get { return EnemyAnimation.FrameWidth; }

}

// Get the height of the enemy ship

public int Height

{

get { return EnemyAnimation.FrameHeight; }

}

// The speed at which the enemy moves

float enemyMoveSpeed;

在Game1类中AddEnemy方法将敌人加载到游戏中。

XNA Project: Shooter File: Game1.cs

private void AddEnemy()

{

// Create the animation object

Animation enemyAnimation = new Animation();

// Initialize the animation with the correct animation information

enemyAnimation.Initialize(enemyTexture, Vector2.Zero, 47, 61, 8, 30, Color.White, 1f, true);

// Randomly generate the position of the enemy

Vector2 position = new Vector2(GraphicsDevice.Viewport.Width + enemyTexture.Width / 2, random.Next(100, GraphicsDevice.Viewport.Height - 100));

// Create an enemy

Enemy enemy = new Enemy();

// Initialize the enemy

enemy.Initialize(enemyAnimation, position);

// Add the enemy to the active enemies list

enemies.Add(enemy);

}

在Game1类的UpdateEnemies方法中更新敌人。如果敌人的Active属性为false，且生命值小于或等于零，则在敌人的位置坐标发生爆炸，播放爆炸的声音效果，增加游戏玩家—英雄飞艇的积分。

XNA Project: Shooter File: Game1.cs

private void UpdateEnemies(GameTime gameTime)

{

// Spawn a new enemy enemy every 1.5 seconds

if (gameTime.TotalGameTime - previousSpawnTime > enemySpawnTime)

{

previousSpawnTime = gameTime.TotalGameTime;

// Add an Enemy

AddEnemy();

}

// Update the Enemies

for (int i = enemies.Count - 1; i >= 0; i--)

{

enemies[i].Update(gameTime);

if (enemies[i].Active == false)

{

// If not active and health <= 0

if (enemies[i].Health <= 0)

{

// Add an explosion

AddExplosion(enemies[i].Position);

// Play the explosion sound

explosionSound.Play();

//Add to the player's score

score += enemies[i].Value;

}

enemies.RemoveAt(i);

}

}

}

与创建英雄—飞艇类的方法相似，在Enemy类的Initialize、Update和Draw方法中设定敌人的属性，包括位置、生命值、移动速度、对英雄--飞艇的破坏值，以及积分。在Game1类的L Update方法中更新敌人的位置，在Draw渲染和绘制敌人。

### 计算碰撞

计算英雄飞艇与敌人碰撞的算法在UpdateCollision方法中，飞艇和敌人都以矩形图片方式表示，碰撞的判断条件是飞艇和敌人的图形是否重叠。当然飞艇发射的导弹可以击溃敌人，导弹与敌人的碰撞判断条件是导弹与敌人的矩形图片是否重叠。

XNA Project: Shooter File: Game1.cs

private void UpdateCollision()

{

// Use the Rectangle's built-in intersect functionto

// determine if two objects are overlapping

Rectangle rectangle1;

Rectangle rectangle2;

// Only create the rectangle once for the player

rectangle1 = new Rectangle((int)player.Position.X,

(int)player.Position.Y,

player.Width,

player.Height);

// Do the collision between the player and the enemies

for (int i = 0; i < enemies.Count; i++)

{

rectangle2 = new Rectangle((int)enemies[i].Position.X,

(int)enemies[i].Position.Y,

enemies[i].Width,

enemies[i].Height);

// Determine if the two objects collided with each

// other

if (rectangle1.Intersects(rectangle2))

{

// Subtract the health from the player based on

// the enemy damage

player.Health -= enemies[i].Damage;

// Since the enemy collided with the player

// destroy it

enemies[i].Health = 0;

// If the player health is less than zero we died

if (player.Health <= 0)

player.Active = false;

}

}

// Projectile vs Enemy Collision

for (int i = 0; i < projectiles.Count; i++)

{

for (int j = 0; j < enemies.Count; j++)

{

// Create the rectangles we need to determine if we collided with each other

rectangle1 = new Rectangle((int)projectiles[i].Position.X -

projectiles[i].Width / 2, (int)projectiles[i].Position.Y -

projectiles[i].Height / 2, projectiles[i].Width, projectiles[i].Height);

rectangle2 = new Rectangle((int)enemies[j].Position.X - enemies[j].Width / 2,

(int)enemies[j].Position.Y - enemies[j].Height / 2,

enemies[j].Width, enemies[j].Height);

// Determine if the two objects collided with each other

if (rectangle1.Intersects(rectangle2))

{

enemies[j].Health -= projectiles[i].Damage;

projectiles[i].Active = false;

}

}

}

}

在Game1的Update方法中，调用UpdateCollision。

XNA Project: Shooter File: Game1.cs

// Update the collision

UpdateCollision();

### 创建飞艇的武器—导弹

在上一节中我们提及了飞艇发射导弹击溃敌人，即导弹与敌人的碰撞计算。本节中我们将创建导弹类Projectile。同时按下SHIFT + ALT + C，在弹出的窗体中选择**[Code]|[Code file]**。

在新创建的Projectile.cs文件中添加引用。

XNA Project: Shooter File: Projectile.cs

using System;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

添加控制导弹类的变量。

XNA Project: Shooter File: Projectile.cs

// Image representing the Projectile

public Texture2D Texture;

// Position of the Projectile relative to the upper left side of the screen

public Vector2 Position;

// State of the Projectile

public bool Active;

// The amount of damage the projectile can inflict to an enemy

public int Damage;

// Represents the viewable boundary of the game

Viewport viewport;

// Get the width of the projectile ship

public int Width

{

get { return Texture.Width; }

}

// Get the height of the projectile ship

public int Height

{

get { return Texture.Height; }

}

// Determines how fast the projectile moves

float projectileMoveSpeed;

与创建敌人的方法相似，在Projectile类的Initialize、Update和Draw方法中设定导弹的属性，包括位置、移动速度、对敌人的破坏值。在Game1的UpdatePlayer方法中调用AddProjectile方法中加载导弹对象，在Game1的Update方法中调用UpdateProjectiles方法更新导弹。

XNA Project: Shooter File: Game1.cs

private void AddProjectile(Vector2 position)

{

Projectile projectile = new Projectile();

projectile.Initialize(GraphicsDevice.Viewport, projectileTexture, position);

projectiles.Add(projectile);

}

private void UpdateProjectiles()

{

// Update the Projectiles

for (int i = projectiles.Count - 1; i >= 0; i--)

{

projectiles[i].Update();

if (projectiles[i].Active == false)

{

projectiles.RemoveAt(i);

}

}

}

### 实现爆炸效果

当飞艇和敌人，或者导弹和敌人碰撞时，设计爆炸的效果体验真实感。在Game1种声明爆炸的图形和动画。

XNA Project: Shooter File: Game1.cs

// Explosion graphics list

Texture2D explosionTexture;

List<Animation> explosions;

在Game1的Initialize方法中初始化。

XNA Project: Shooter File: Game1.cs

// Initialize the explosion list

explosions = new List<Animation>();

在Game1的LoadContent方法加载爆炸动画。

XNA Project: Shooter File: Game1.cs

explosionTexture = Content.Load<Texture2D>("explosion");

与Projectile类的Add Projectile方法类似，在Game1中添加AddExplosion方法。AddExplosion方法在敌人的生命值小于或等于零时被调用。即在计算碰撞时判断敌人的生命值小于或等于零时被调用。

XNA Project: Shooter File: Game1.cs

private void AddExplosion(Vector2 position)

{

Animation explosion = new Animation();

explosion.Initialize(explosionTexture, position, 134, 134, 12, 45, Color.White, 1f, false);

explosions.Add(explosion);

}

与Projectile类的UpdateProjectiles方法类似，在Game1的Update方法中调用UpdateExplosions方法更新爆炸。

### 游戏音乐

添加音乐和声音，使得游戏更加引人入胜。在Game1类中声明游戏音乐的控制变量。

XNA Project: Shooter File: Game1.cs

// The sound that is played when a laser is fired

SoundEffect laserSound;

// The sound used when the player or an enemy dies

SoundEffect explosionSound;

// The music played during gameplay

Song gameplayMusic;

在Game1的LoadContent方法中加载音乐。

XNA Project: Shooter File: Game1.cs

// Load the music

gameplayMusic = Content.Load<Song>("sound/gameMusic");

// Load the laser and explosion sound effect

laserSound = Content.Load<SoundEffect>("sound/laserFire");

explosionSound = Content.Load<SoundEffect>("sound/explosion");

在Game1的PlayMusic方法中调用MediaPlayer的Play方法播放音乐。

XNA Project: Shooter File: Game1.cs

private void PlayMusic(Song song)

{

// Due to the way the MediaPlayer plays music,

// we have to catch the exception. Music will play when the game is not tethered

try

{

// Play the music

MediaPlayer.Play(song);

// Loop the currently playing song

MediaPlayer.IsRepeating = true;

}

catch { }

}

在Game1的UpdatePlayer方法中调用AddProjectile即发射导弹后，播放激光的声音。

XNA Project: Shooter File: Game1.cs

// Add the projectile, but add it to the front and center of the player

AddProjectile(player.Position + new Vector2(player.Width / 2, 0));

// Play the laser sound

laserSound.Play();

在Game1类的UpdateEnemies方法中AddExplosion即爆炸发生时，播放爆炸的声音。

XNA Project: Shooter File: Game1.cs

// Add an explosion

AddExplosion(enemies[i].Position);

// Play the explosion sound

explosionSound.Play();

### 在模拟器中运行

按F5运行应用程序，或者点击Start Debugging按钮运行，如图16-2 Start Debugging。



图16-2 Start Debugging

图16-3 飞艇游戏