# 第12章 Angry Panda游戏主场景的实现

第11章我们共同了解了示例游戏Angry Panda的设计和框架构建，并依次实现了游戏中的数据管理和常量定义、音效管理、游戏的开始界面、游戏介绍界面、关卡选择界面、游戏的关卡结果显示界面、用于创建定制动画的类。

本章将学习如何使用Box2D物理引擎创建游戏中的各类物体、构建游戏的主场景、实现游戏的逻辑机制，并使用碰撞检测机制来完成最终的互动。限于篇幅，文中不会列出所有的源代码，请读者参考示例项目中的完整源代码。

## Box2D世界的物体

考虑到游戏的扩展性，我们不会直接在主场景中添加创建物体的代码，而是定义若干个类专门用于创建场景中的物体。此外，为了更好地模拟现实中的物理世界，这里将采用Box2D物理引擎来实现物体的创建和互动。在这一部分内容中，将创建以下类：

* 抽象类PhysicsNode
* Panda类（熊猫）
* Enemy类（外星怪物）
* FirePlatform类（弹弓基座）
* GroundPlane类（地面物体）
* StackObject类（遮挡物体）
* ShootingTargets类（攻击的物体）

开发者还可以根据自己的需要创建更多物体类，从而对游戏进行扩展。

### 创建抽象物体类

为了方便创建和管理游戏中的物体，首先创建一个抽象类PhysicsNode，该类直接继承自CCNode类，是CCNode类和b2Body类的拓展。其中的实例方法分别用于使用精灵帧来创建物体、删除物体、删除精灵、淡出并消失、让物体成为静态、删除精灵和物体。

在Xcode中右键单击AngryPanda的项目名称，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为PhysicsNode.mm，单击Create。

在Xcode中切换到PhysicsNode.h，替换其中的代码如代码清单12-1所示。

1. PhysicsNode.h文件

#import <Foundation/Foundation.h>

#import "cocos2d.h"

#import "Box2D.h"

#import "Constants.h"

@interface PhysicsNode : CCNode {

b2Body\* body;

CCSprite\* sprite;

}

@property (readonly, nonatomic) b2Body\* body;

@property (readonly, nonatomic) CCSprite\* sprite;

-(void) createBodyWithSpriteAndFixture:(b2World\*)world bodyDef:(b2BodyDef\*)bodyDef fixtureDef:(b2FixtureDef\*)fixtureDef spriteName:(NSString\*)spriteName;

-(void) removeBody;

-(void) removeSprite;

-(void) removeSpriteAndBody;

-(void) fadeThenRemove;

-(void) makeBodyStatic;

-(void) makeBodyDynamic;

@end

从上面的代码看到，PhysicsNode类继承自CCNode，在其变量声明部分声明了两个变量，分别是到b2Body物体的引用，以及到CCSprite精灵的引用。

在属性声明之后声明了若干个实例方法，打开示例项目，并将项目中PhysicsNode.mm中的代码复制到所创建的项目文件中。接下来将对其中的重要方法加以解释。

#### createBodyWithSpriteAndFixture方法

该方法是PhysicsNode类最重要的方法，作用是使用精灵帧图片来创建物体，其实现代码如代码清单12-2所示。

1. createBodyWithSpriteAndFixture方法的实现

-(void) createBodyWithSpriteAndFixture:(b2World\*)world bodyDef:(b2BodyDef\*)bodyDef fixtureDef:(b2FixtureDef\*)fixtureDef spriteName:(NSString\*)spriteName {

[self removeBody]; //if remove the body if it already exists

[self removeSprite]; //if remove the sprite if it already exists

sprite = [CCSprite spriteWithFile:spriteName];

[self addChild:sprite];

body = world->CreateBody(bodyDef);

body->SetUserData(self);

if ( fixtureDef != NULL)

{

body->CreateFixture(fixtureDef);

}

}

在了解这段代码的作用之前，先回顾Box2D世界中创建物体的步骤：

1. 创建一个物体定义，并使用物体定义来指定物体的初始属性，比如位置和速度等。
2. 使用世界对象的物体工厂来创建物体。
3. 为物体定义一个形状（shape），代表要模拟的物体的几何形状。
4. 创建一个夹具定义，并将夹具定义的形状设置为刚刚所定义的形状，同时还可以在这里设置一些其他的属性，比如密度和表面摩擦力等。对于地面物体来说，这一步通常可以省略。
5. 使用物体的夹具工厂来创建夹具，必要时还可以保存到该夹具的引用，以便后续使用。由于后面不需要引用地面夹具，因此这里就没有创建相关的引用。

注意 可以给单个物体添加多个夹具，如地面物体。当创建复杂对象的时候，这一点会很有用。

如代码清单12-2所示，首先调用removeBody方法和removeSprite方法清除已有的物体和精灵。然后根据所传入的spriteName参数创建一个精灵，并添加为当前层的子节点。接下来根据传入的bodyDef物体定义，使用世界对象的物体工厂创建物体。然后使用SetUserData设置物体所对应的用户自定义数据为自身。此时，如果所传入的fixtureDef夹具定义不是NULL，则使用物体的夹具工厂创建夹具。

接下来的几个方法都相对比较简单，它们的作用及作用方法如下所示。

#### makeBodyStatic方法

将物体类型设置为静态物体，相关代码如下：

body->SetType(b2\_staticBody);

#### makeBodyDynamic方法

将物体类型设置为动态物体，相关代码如下：

body->SetType(b2\_ dynamicBody);

#### fadeThenRemove方法

执行序列动作，让精灵和物体淡出并消失，其相关代码如下：

CCSequence \*seq = [CCSequence actions:

[CCFadeTo actionWithDuration:1.0f opacity:0],

[CCCallFunc actionWithTarget:self selector:@selector(removeSpriteAndBody)], nil];

[[self sprite] runAction:seq];

在以上代码中，首先定义一个CCSequence序列动作，其中第一个动作是在1.0秒内让物体变得完全透明；第二个动作是调用removeSpriteAndBody方法，即清除当前的精灵和物体；最后获取PhysicsNode的精灵对象，并让其执行该序列动作。

#### removeBody方法

从Box2D世界中删除物体，相关代码如下：

if( body != NULL)

{

body->GetWorld()->DestroyBody(body);

body = NULL;

}

在以上代码中，首先判断当前物体是否为NULL，如果不是，则使用GetWorld()方法获取物体所在的Box2D世界，然后使用DestroyBody(body)方法删除物体，最后将物体设置为NULL。

#### removeSprite方法

从Cocos2D世界中删除精灵对象，相关代码如下：

if (sprite != nil) {

[self removeAllChildrenWithCleanup:YES];

sprite = nil;

}

以上代码首先判断精灵对象是否为nil，如果不是，调用removeAllChildrenWithCleanup方法清除精灵对象及其所有字节点，最后将精灵对象设置为nil。

#### removeSpriteAndBody方法

同时删除精灵对象和物体，其作用相当于同时调用removeBody方法和removeSprite方法。相关代码如下：

if (sprite != nil) {

[self removeAllChildrenWithCleanup:YES];

sprite = nil;

}

if( body != NULL)

{

body->GetWorld()->DestroyBody(body);

body = NULL;

}

创建PhysicsNode抽象类之后，就可以使用该类的继承创建具体的物体。

### 创建熊猫角色

为了在游戏场景中添加熊猫，需要单独创建一个Panda类，用于创建和管理游戏中的熊猫角色，其效果如图12-1所示。



1. 熊猫角色

该类将在ShootingTargets类中创建，它继承自PhysicsNode，其中所定义的几个实例方法分别用于在Box2D中初始化并创建熊猫、创建熊猫、显示在空中的状态、在翻滚中的状态、在地面的状态、在弹弓中的状态、站立时的状态等。

在Xcode中右键单击AngryPanda的项目名称，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为Panda.mm，单击Create。

在Xcode中切换到Panda.h，并替换其中的代码如代码清单12-3所示。

1. Panda.h文件内容

#import "PhysicsNode.h"

@interface Panda : PhysicsNode {

b2World\* theWorld;

NSString\* baseImageName;

NSString\* spriteImageName;

CGPoint initialLocation;

bool onGround;

unsigned char counter;

}

@property (nonatomic) bool onGround;

+(id) pandaWithWorld:(b2World\*)world location:(CGPoint)p baseFileName:(NSString\*)name;

-(id) initWithWorld:(b2World\*)world location:(CGPoint)p baseFileName:(NSString\*)name;

-(void) createPanda;

-(void) spriteInAirState;

-(void) spriteInRollState;

-(void) spriteInRollStateWithAnimationFirst;

-(void) spriteInGroundState;

-(void) spriteInSlingState;

-(void) spriteInStandingState;

-(void) spriteInPulledBackSlingState;

@end

从以上代码中可以看到，Panda类直接继承自PhysicsNode类，但同时也声明了自己的变量和实例方法。

接下来看看实例方法的具体内容和作用。

#### +(id)pandaWithWorld类方法

调用initWithWorld这个实例方法来创建并初始化一个熊猫物体。

initWithWorld方法的作用是设置所在的Box2D物理世界，初始位置，精灵帧图片名称等参数，然后调用createPanda方法来创建熊猫物体。其中world代表所在的Box2D物理世界,location代表熊猫的初始位置，baseFileName代表精灵帧图的名称。

#### createPanda方法

该方法是Panda类的核心方法，用来创建一个熊猫物体，其实现代码如代码清单12-4所示。

1. createPanda方法的实现

-(void) createPanda {

spriteImageName = [NSString stringWithFormat:@"%@\_standing.png", baseImageName];

onGround = NO;

// 1.创建物体定义

b2BodyDef bodyDef;

bodyDef.type = b2\_staticBody;

bodyDef.position.Set(initialLocation.x/PTM\_RATIO, initialLocation.y/PTM\_RATIO);

//2.定义形状

b2CircleShape shape;

float radiusInMeters = (40 / PTM\_RATIO) \* 0.5f; //对于不同的圆形大小，可以加上或减去40

shape.m\_radius = radiusInMeters;

// 3.创建物体夹具

b2FixtureDef fixtureDef;

fixtureDef.shape = &shape;

fixtureDef.density = 1.0f;

fixtureDef.friction = 1.0f;

fixtureDef.restitution = 0.1;

[super createBodyWithSpriteAndFixture:theWorld bodyDef:&bodyDef fixtureDef:&fixtureDef spriteName:spriteImageName];

}

在以上代码中，首先使用所传入的参数确定精灵帧图片的文件名，然后将onGround数属性设置为NO。接下来是创建Box2D物体的具体步骤：

1. 创建物体定义，并设置相关属性。这里将物体设置为静态物体，并设置了所在的初始位置。
2. 定义形状，并设置圆形半径。
3. 创建物体夹具定义，并设置夹具的形状为刚才所定义的形状。设置密度，摩擦力和弹性系数等参数。

使用super调用父类(PhysicsNode类)的createBodyWithSpriteAndFixture方法创建物体。

+(id)pandaWithWorld类方法的作用是调用initWithWorld实例方法来创建并初始化一个熊猫物体。

spriteInSlingState、spriteInStandingState、spriteInAirState、spriteInRollState、spriteInPulledBackSlingState、spriteInGroundState方法的作用类似，都使用CCNode的setTexture方法设置了灵对象在不同状态下所显示的图片。

#### spriteInRollStateWithAnimationFirst方法

每帧执行rollAnimation动画，相关的代码如下：

[self schedule:@selector(rollAnimation:) interval:1.0f/60.0f];

以上代码预定了一个rollAnimation消息，并每帧执行一次。

#### rollAnimation方法

使用setTexture方法，每帧显示不同的精灵帧图片。

可以看到，这里使用了一种全新的方式创建和显示精灵帧动画，即使用消息预定加上setTexture的方式。在第3章曾经接触过另外两种创建和显示动画的方式，这里简单回顾：

* 1. **使用精灵帧图创建动画**

1. 创建精灵对象并将其添加为当前层的子节点

CGSize size = [CCDirector sharedDirector].winSize;

CCSprite \*mySprite = [CCSprite spriteWithFile:@"pandawalk1.png"];

mySprite.position = ccp(size.width/2,size.height/2);

[self addChild:mySprite];

1. 创建CCAnimation动画，指定动画帧的内容

CCAnimation \*anim = [CCAnimation animation];

[anim addFrameWithFilename:@"pandawalk1.png"];

[anim addFrameWithFilename:@"pandawalk2.png"];

[anim addFrameWithFilename:@"pandawalk3.png"];

1. 创建CCAnimate动画动作，并让精灵对象执行。通过指定restoreOriginalFrame为YES，可以让精灵对象在动画执行完成后恢复到最初的状态

id animAction = [CCAnimate actionWithDuration:0.5f animation:anim restoreOriginalFrame:YES];

id repeatanimAction = [CCRepeatForever actionWithAction:animAction];

[mySprite runAction:repeatanimAction];

在以上代码中，首先创建精灵对象并添加为当前层的子节点；然后创建CCAnimation动画，并指定动画帧的内容；最后创建CCAnimate动画动作，并让精灵对象执行。

* 1. **使用精灵表单创建动画**

1. 使用Plist文件将精灵帧纹理添加到精灵帧缓存中

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@"AnimPanda.plist"];

1. 创建一个CCSpriteBatchNode精灵表单

CCSpriteBatchNode \*batchNode = [CCSpriteBatchNode batchNodeWithFile:@"AnimPanda.png"];

[self addChild:batchNode];

1. 创建精灵帧列表

NSMutableArray \*animFrames = [NSMutableArray array];

for(int i=1; i<3;i++){

[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]spriteFrameByName:[NSString stringWithFormat:@"pandawalk%d.png",i]]];

}

1. 创建动画对象

CCAnimation \*walkAnim = [CCAnimation animationWithFrames:walkAnimFrames delay:0.1f];

（5）创建精灵，并运行动画

CGSize size = [CCDirector sharedDirector].winSize;

CCSprite \*panda = [CCSprite spriteWithSpriteFrameName:@"pandawalk1.png"];

panda.position = ccp(size.width\*0.8,size.height\*0.4);

id walkAction = [CCRepeatForever actionWithAction:[CCAnimate actionWithAnimation:walkAnim restoreOriginalFrame:YES]];

[panda runAction:walkAction];

[panda runAction:[CCMoveTo actionWithDuration:6.0f position:ccp(size.width\*0.2,size.height\*0.4)]];

[batchNode addChild:panda];

注意 我们已经学会了三种创建精灵帧动画的方式。

第一种方式，即本示例项目中所使用的方式可以方便在多个类中重用。

第二种方式，即使用精灵帧图片来创建动画的代码相对比较简单。

第三种方式，即使用精灵表单来创建动画的方式则是在运行效率上更胜一筹。

至于开发者在实际项目中采用哪一种动画创建方式，取决于实际项目的需求。

### 创建平台物体

看懂了Panda类的实现代码后，会发现FirePlatform和GroundPlane类非常简单，其中FirePlatform类主要用于创建抛投熊猫所使用的平台物体，如图12-2所示。而GroundPlane类则用于创建地平面物体。



1. 抛投熊猫所使用的平台物体

在Xcode中右键单击AngryPanda的项目名称，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为FirePlatform.mm，单击Create。

在Xcode中切换到FirePlatform.h，并替换其中的代码如代码清单12-5所示。

1. FirePlatform.h文件

#import "PhysicsNode.h"

@interface FirePlatform : PhysicsNode {

b2World\* theWorld;

NSString\* spriteImageName;

CGPoint initialLocation;

}

+(id) platformWithWorld:(b2World\*)world location:(CGPoint)location spriteFileName:(NSString\*)spriteFileName;

-(id) initWithWorld:(b2World\*)world location:(CGPoint)location spriteFileName:(NSString\*)spriteFileName;

-(void) createPlatform;

@end

从以上代码中可以看到，FirePlatform类直接继承自PhysicsNode类，并添加了自己的变量和实例方法。

接下来具体看看实例方法的内容和作用：

#### +(id) platformWithWorld类方法

调用initWithWorld实例方法来创建并初始化一个平台物体。

#### initWithWorld方法

设置所在的Box2D物理世界、初始位置、精灵帧图片名称等参数，然后调用createPlatform方法来创建平台物体。

#### createPlatform方法

具体创建平台物体，其实现代码和Panda类中创建熊猫物体的代码类似，唯一的区别在于形状定义部分，其相关代码如下：

b2PolygonShape shape;

int num = 4;

b2Vec2 vertices[] = {

b2Vec2(-102.0f / PTM\_RATIO, -49.5f / PTM\_RATIO),

b2Vec2(-113.0f / PTM\_RATIO, -81.5f / PTM\_RATIO),

b2Vec2(113.0f / PTM\_RATIO, -84.5f / PTM\_RATIO),

b2Vec2(106.0f / PTM\_RATIO, -47.5f / PTM\_RATIO)

};

shape.Set(vertices, num);

在以上代码首先定义一个b2PolygonShape多边形，设置多边形的顶点数量为4，并使用b2Vec2 创建了一个顶点坐标数组，最后使用shape.Set(vertices,num)确定了该多边形。

此外需要注意，这里添加了两个多边形，即该弹弓基座共创建了两个夹具。

GroundPlane类的实现代码和FirePlatform类几乎没有任何区别，这里不再过多解释，开发者可以自行查看示例项目中的相关代码。

### 创建外星怪物

相对于刚才的熊猫物体和平台物体，代表外星怪物的Enemy类要稍微复杂一些。

该类将在ShootingTargets类中创建，它继承自PhysicsNode，其中所定义的几个实例方法作用分别是在Box2D世界中初始化并创建敌人、让敌人受到伤害、让敌人分解等。

在Xcode中右键单击AngryPanda的项目名称，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为Enemy.mm，单击Create。

在Xcode中切换到Enemy.h，并替换其中的代码如代码清单12-6所示。

1. Enemy.h文件

#import "PhysicsNode.h"

#import "GameSounds.h"

@interface Enemy : PhysicsNode {

b2World\* theWorld;

NSString\* spriteImageName;

NSString\* baseImageName;

CGPoint initialLocation;

int breakAfterHowMuchContact; //如果为0，则在首次接触后将敌人分解

int damageLevel; //记录已完成的伤害程度

bool breakOnNextDamage;

bool isRotationFixed; //如果设置为YES，则外星怪物不会旋转

float theDensity;

unsigned char shapeCreationMethod; //和所有遮挡物体相同， 在Constants.h中检查形状定义

bool damageFromGroundContact;

bool damageFromDamageEnabledStackObjects; //遮挡物被启用可以伤害敌人

bool differentSpritesForDamage; //是否已在伤害过程中包含不同的图片

int pointValue;

int simpleScore; //在constants中定义，当敌人分解时使用何种视觉效果

int currentFrame;

int framesToAnimateOnBreak; //如果设置为0，则不显示任何分解动画帧

bool enemyCannotBeDamagedForShortInterval; // 在伤害发生后，外星怪物会获得一定时间的“无敌”状态

}

@property (nonatomic) bool damageFromGroundContact;

@property (nonatomic) bool damageFromDamageEnabledStackObjects;

@property (nonatomic) bool breakOnNextDamage;

@property (nonatomic) int pointValue;

@property (nonatomic) int simpleScore;

+(id) enemyWithWorld:(b2World\*)world location:(CGPoint)location spriteFileName:(NSString\*)spriteFileName isTheRotationFixed:(bool)isTheRotationFixed getDamageFromGround:(BOOL)getDamageFromGround doesGetDamageFromDamageEnabledStackObjects:(BOOL)doesGetDamageFromDamageEnabledStackObjects breaksFromHowMuchContact:(int)breaksFromHowMuchContact hasDifferentSpritesForDamage:(bool)hasDifferentSpritesForDamage numberOfFramesToAnimateOnBreak:(int)numberOfFramesToAnimateOnBreak density:(float)density createHow:(int)createHow points:(int)points simpleScoreType:(int)simpleScoreType;

-(id) initWithWorld:(b2World\*)world location:(CGPoint)location spriteFileName:(NSString\*)spriteFileName isTheRotationFixed:(bool)isTheRotationFixed getDamageFromGround:(BOOL)getDamageFromGround doesGetDamageFromDamageEnabledStackObjects:(BOOL)doesGetDamageFromDamageEnabledStackObjects breaksFromHowMuchContact:(int)breaksFromHowMuchContact hasDifferentSpritesForDamage:(bool)hasDifferentSpritesForDamage numberOfFramesToAnimateOnBreak:(int)numberOfFramesToAnimateOnBreak density:(float)density createHow:(int)createHow points:(int)points simpleScoreType:(int)simpleScoreType;

-(void) createEnemy;

-(void) damageEnemy;

-(void) breakEnemy;

-(void) makeUnScoreable;

@end

在以上代码中，首先在头文件中导入了PhysicsNode和GameSounds这两个类的头文件。Enemy类仍然继承自PhysicsNode，并声明了多个实例变量和方法。

接下来将对该类的一些重要方法加以解释：

#### +(id) enemyWithWorld类方法

调用initWithWorld实例方法创建并初始化外星怪物。

#### initWithWorld方法

设置所在的Box2D物理世界、初始位置、精灵帧图片名称、伤害值等参数，然后调用createEnemy方法创建外星怪物。其中各参数的含义如下：

* world：所在的Box2D物理世界。
* location：外星怪物的位置。
* spriteFileName：所使用的精灵帧图。
* isTheRotationFixed：是否可以旋转。
* getDamageFromGround：是否怪物在碰到地面时会受到伤害。
* doesGetDamageFromDamageEnabledStackObjects：怪物在碰到屏障物体时是否会受到伤害。
* breaksFromHowMuchContact：几次碰撞后才会分解（如果是0，则一次碰撞后就会消失）。
* hasDifferentSpritesForDamage：怪物是否有受伤害的动画帧图。
* numberOfFramesToAnimateOnBreak：怪物分解的动画帧数。
* createHow：创建怪物的形状类型。
* simpleScoreType：怪物分解时的视觉效果类型。

#### createEnemy方法

在Box2D物理世界中具体创建外星怪物。与之前创建熊猫和平台物体不同，考虑到游戏的可扩展性，在形状设置方面根据shapeCreationMethod参数的不同，而设置不同的形状。以最后一种参数为例说明：

else if ( shapeCreationMethod == customCoordinates1) { //use your own custom coordinates from a program like Vertex Helper Pro

int num = 4;

b2Vec2 vertices[] = {

b2Vec2(-64.0f / PTM\_RATIO, 16.0f / PTM\_RATIO),

b2Vec2(-64.0f / PTM\_RATIO, -16.0f / PTM\_RATIO),

b2Vec2(64.0f / PTM\_RATIO, -16.0f / PTM\_RATIO),

b2Vec2(64.0f / PTM\_RATIO, 16.0f / PTM\_RATIO)

};

shape.Set(vertices, num);

}

当形状创建方法是定制化顶点坐标时（由开发者根据需要决定），首先定义顶点的数量（通常不超过7个），然后设置b2Vec数组，将自己从诸如Vertex Helper Pro等软件中获取的精灵图片的顶点坐标添加到该数组中，最后使用shape.Set(vertices,num)创建一个完全定制化的形状。

#### damageEnemy方法

当怪物被命中时执行相关的操作，其相关代码如下：

if ( enemyCannotBeDamagedForShortInterval == NO ) {

damageLevel ++;

enemyCannotBeDamagedForShortInterval = YES;

[self performSelector:@selector(enemyCanBeDamagedAgain) withObject:nil afterDelay:1.0f];

if ( differentSpritesForDamage == YES) {

[sprite setTexture:[[CCSprite spriteWithFile:[NSString stringWithFormat:@"%@\_damage%i.png", baseImageName, damageLevel]] texture] ];

}

if ( damageLevel == breakAfterHowMuchContact ) {

breakOnNextDamage = YES;

}

}

在以上代码中，首先判断怪物是否短时间内可以被伤害，如果是则执行下面的代码。首先将受伤级别加1，然后设置怪物短时间内不可以再次被伤害，在延迟1秒后执行enemyCanBeDamagedAgain方法。

接下来判断是否存在怪物受伤害后的动画帧图，如果是则调用setTexture方法创建怪物受伤害的动画。最后判断受伤级别与breakAfterHowMuchContact（在几次碰撞后分解）的关系，如果相同，则怪物将在下一次受伤时分解。

#### breakEnemy方法

预定消息，每两帧执行一次分解动画，即调用startBreakAnimation方法。

#### startBreakAnimation方法

使用setTexture方法创建怪物分解的动画帧图。

### 创建遮挡物体

StackObject类用于创建并管理游戏中遮挡怪物的屏障物体，类似Angry Birds中的木头，玻璃等。该类将在ShootingTargets类中创建，它继承自PhysicsNode，其中所定义的几个实例方法作用分别是在Box2D世界中初始化并创建遮挡物、创建物体、启动分解动画、当与地面接触时播放分解动画、当与熊猫接触时播放分解动画等。

在Xcode中右键单击AngryPanda的项目名称，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为StackObject.mm，单击Create。

在Xcode中切换到StackObject.h，并替换其中的代码如代码清单12-7所示。

1. StackObject.h文件

#import "PhysicsNode.h"

#import "GameSounds.h"

@interface StackObject : PhysicsNode {

b2World\* theWorld;

NSString\* spriteImageName;

NSString\* baseImageName;

CGPoint initialLocation;

bool addedAnimatedBreakFrames;

bool breakOnGroundContact;

bool breakOnPandaContact;

bool canDamageEnemy;

bool isStatic; //是否是静态物体

float theDensity;

unsigned char shapeCreationMethod;

int angle;

int currentFrame;

int framesToAnimate;

int pointValue;

int simpleScoreVisualFX; //defined in Constants.h

}

@property (nonatomic) bool isStatic;

@property (nonatomic) bool breakOnGroundContact;

@property (nonatomic) bool canDamageEnemy;

@property (nonatomic) int pointValue;

@property (nonatomic) int simpleScore;

+(id) objectWithWorld:(b2World\*)world location:(CGPoint)location spriteFileName:(NSString\*)spriteFileName breakOnGround:(BOOL)breakOnGround breakFromPanda:(BOOL)breakFromPanda hasAnimatedBreakFrames:(bool)hasAnimatedBreakFrames damageEnemy:(BOOL)damageEnemy density:(float)density createHow:(int)createHow angleChange:(int)angleChange makeImmovable:(bool)makeImmovable points:(int)points simpleScoreType:(int)simpleScoreType;

-(id) initWithWorld:(b2World\*)world location:(CGPoint)location spriteFileName:(NSString\*)spriteFileName breakOnGround:(BOOL)breakOnGround breakFromPanda:(BOOL)breakFromPanda hasAnimatedBreakFrames:(bool)hasAnimatedBreakFrames damageEnemy:(BOOL)damageEnemy density:(float)density createHow:(int)createHow angleChange:(int)angleChange makeImmovable:(bool)makeImmovable points:(int)points simpleScoreType:(int)simpleScoreType;

-(void) createObject;

-(void) startBreakAnimation:(ccTime) delta;

-(void) playBreakAnimationFromGroundContact;

-(void) playBreakAnimationFromPandaContact ;

-(void) makeUnScoreable;

@end

接下来将对该类的一些重要方法加以解释：

#### objectWithWorld类方法

作用是调用initWithWorld实例方法创建并初始化屏障物体。initWithWorld方法中设置多个参数，并调用createObject方法创建屏障物体。这里要对各个参数的作用加以简要解释。如下所示：

* world：物体所在的Box2D物理世界。
* location：物体所放置的位置。
* spriteFileName：物体所对应的精灵帧图。
* breakOnGround：该布尔变量代表屏障物体是否会在碰到地面时分解。
* breakFromPanda：该布尔变量代表屏障物体是否会在碰到熊猫时分解。
* hasAnimatedBreakFrames：该布尔变量代表是否有物体分解的动画帧图。
* damageEnemy：该布尔变量代表屏障物体是否会伤害到所碰到的外星怪物。
* density：物体的密度。
* createHow：所使用的形状类型。
* angelChange：物体的旋转角度。
* makeImmovable：物体是否可以移动。
* points：碰撞时的得分。
* simpleScoreType：屏障物体分解时的视觉效果。

#### createObject方法

该方法和Enemy类中的createEnemy方法类似，根据所选择的shapeCreateMethod（形状类型）参数创建屏障物体。

#### playBreakAnimationFromPandaContact方法

当屏障物体碰到熊猫时，如果breakOnPandaContact属性为YES，则预定消息。每两帧运行一次startBreakAnimation方法以播放屏障物体分解的动画。

#### startBreakAnimation方法

使用setTexture方法创建怪物分解的动画帧图。

至此，熊猫要攻击的屏障和外星怪物都已就绪，接下来使用ShootingTargets类具体放置熊猫要攻击的目标。

### 创建并放置攻击目标

创建ShootingTargets类，然后调用之前创建的Enemy类和StackObject类，从而在游戏的主场景中创建和放置屏障物体和外星怪物。该类将在MainScene类中调用。

在Xcode中右键单击AngryPanda的项目名称，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为ShootingTargets.mm，单击Create。

在Xcode中切换到ShootingTargets.h，并替换其中的代码如代码清单12-8所示。

1. ShootingTargets.h文件

#import <Foundation/Foundation.h>

#import "cocos2d.h"

#import "Box2D.h"

@interface ShootingTargets : CCNode {

b2World\* world;

int currentLevelNumber;

int stackLocationX; //stack在x轴上的起始位置

int stackLocationY; //stack在y轴上的起始位置

int stackAdjustmentX; //根据关卡进行调整

int stackAdjustmentY; //根据关卡进行调整

}

+(id) setupStackWithWorld:(b2World\*)theWorld;

@end

从以上代码中可以看到，ShootingTargets类继承自CCNode，而非PhysicsNode类。

接下来将对该类的一些重要方法加以解释：

#### setupStackWithWorld类方法

调用initStackWithWorld实例方法创建并初始化熊猫要攻击的物体。

#### initStackWithWorld方法

该方法的作用则是根据当前关卡编号调用不同的方法，以在场景中放置熊猫要攻击的物体，其相关代码如代码清单12-9所示。

1. initStackWithWorld方法的实现

if (IS\_IPAD) {

stackLocationX = 1400; //整个stack在iPad中x轴上的起始位置

stackLocationY = 100; //整个stack在iPad中y轴上的起始位置

} else if(IS\_IPHONE){

stackLocationX = 900; //整个stack在iPhone中x轴上的起始位置

stackLocationY = 35; //整个stack在iPhone中y轴上的起始位置

}

currentLevelNumber = [GameData sharedData].currentLevel;

switch (currentLevelNumber) {

case 1:

[self performSelector:@selector(createLevel1)];

break;

case 2:

[self performSelector:@selector(createLevel2)];

break;

default:

[self performSelector:@selector(createLevel2)];

break;

}

以上代码首先根据设备的不同设置攻击目标在屏幕中的摆放位置。接下来使用GameData的单例获取当前关卡编号。然后根据当前关卡编号使用performSelector调用不同的方法。

#### createLevel1方法

createLevel1和createLevel2这两个方法是ShootingTargets类的核心方法。当然，根据游戏的需要，玩家还可以添加createLevel3、 createLevel4等更多的场景摆放方法。以createLevel1为例看看该方法的实现，如代码清单12-10所示。

1. createLevel1方法的实现

-(void) createLevel1 {

if (IS\_IPAD) {

stackAdjustmentX = 0;

stackLocationX = stackLocationX - stackAdjustmentX;

stackAdjustmentY = 0;

stackLocationY = stackLocationY - stackAdjustmentY;

} else {

stackAdjustmentX = 0;

stackLocationX = stackLocationX - stackAdjustmentX;

stackAdjustmentY = 0;

stackLocationY = stackLocationY - stackAdjustmentY;

}

StackObject\* object1 = [StackObject objectWithWorld:world location:ccp( 0 + stackLocationX , 65 + stackLocationY) spriteFileName:@"woodShape1" breakOnGround:NO breakFromPanda:YES hasAnimatedBreakFrames:YES damageEnemy:NO density:0.25f createHow:useShapeOfSourceImage angleChange:90 makeImmovable:NO points:100 simpleScoreType:breakEffectSmokePuffs];

[self addChild:object1 z:zOrderStack];

// ……

//此处省略了创建其他物体的部分代码，具体请查看示例项目中的源代码

// ……

Enemy\* enemy1 = [Enemy enemyWithWorld:world location:ccp(45 + stackLocationX , 200 + stackLocationY) spriteFileName:@"dragon" isTheRotationFixed:YES getDamageFromGround:YES doesGetDamageFromDamageEnabledStackObjects:YES breaksFromHowMuchContact:0 hasDifferentSpritesForDamage:YES numberOfFramesToAnimateOnBreak:10 density:1.0f createHow:useShapeOfSourceImageButSlightlySmaller points:10000 simpleScoreType:breakEffectSmokePuffs];

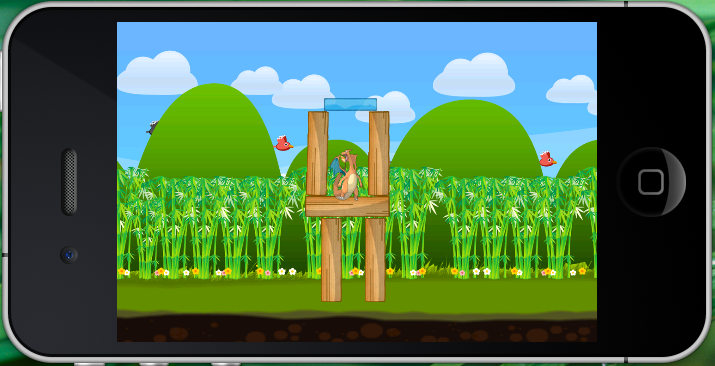
[self addChild:enemy1 z:zOrderStack];

}

以上代码内容看起来很多，但实际上很容易理解。首先根据设备的不同设置摆放位置的微调。然后调用StackObject的类方法objectWithWorld，并指定相关的参数创建了6个屏障物体。最后调用Enemy的类方法enemyWithWorld，并指定相关的参数创建了1个外星怪物。

提示 在游戏场景中，创建屏障物体和外星怪物的关键是如何指定其所在的位置。开发者可以使用Photoshop或Flash等软件，将要摆放的屏障和怪物图片放到一个空白的场景（和iPad或iPhone设备的大小相同）中，然后打开绘图软件的标尺功能，并记下屏障物体和怪物所在的坐标即可。

外星怪物及屏障物体如图12-3所示。



1. 游戏中的外星怪物及屏障物体

至此，已经准备好游戏主场景中的所有物体，接下来实现游戏的核心部分MainScene类。

## 实现游戏的主场景

场景中的物体、音效、数据管理、各类动画和常量都准备完毕后，就可以实现游戏的关卡主场景了。游戏的关卡主场景是一款游戏的核心所在。之前所提到的各个类都可以轻松重用到其他项目中，而主场景则往往只能被特定的项目所独享。本节讲解如何模仿Angry Birds实现游戏的主场景。最终实现的效果如图12-4所示。



1. 初次进入关卡1的游戏主场景
2. 创建MainScene类

在Xcode中右键单击AngryPanda的项目名称，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为MainScene.mm，单击Create。

在示例项目中找到MainScene.h和MainScene.mm两个文件，并将其中的代码拷贝粘贴到刚才所创建的文件中。

在Xcode中首先切换到MainScene.h，如代码清单12-11所示。

1. MainScene.h文件

#import "cocos2d.h"

#import "Box2D.h"

#import "GLES-Render.h"

#import "Constants.h"

#import "PhysicsNode.h"

#import "Panda.h"

#import "FirePlatform.h"

#import "GroundPlane.h"

#import "ContactListener.h"

#import "StackObject.h"

#import "GameData.h"

#import "ShootingTargets.h"

#import "LevelSelection.h"

#import "GameSounds.h"

#import "GamePause.h"

// MainScene

@interface MainScene : CCLayer

{

int screenWidth;

int screenHeight;

int currentLevel;

int pointTotalThisRound;

int pointsToPassLevel;

int bonusThisRound;

int bonusPerExtraPanda;

float panAmount;

int initialPanAmount;

int extraAmountOnPanBack;

int maxStretchOfSlingShot;

BOOL stackIsNowDynamic;

BOOL areWeOnTheIPad;

BOOL useImagesForPointScoreLabels;

BOOL somethingJustScored;

BOOL dottingOn;

BOOL areWeInTheStartingPosition;

BOOL slingShotPandaInHand;

BOOL throwInProgress;

BOOL autoPanningInProgress;

BOOL reverseHowFingerPansScreen;

BOOL panningTowardSling;

BOOL continuePanningScreenOnFingerRelease;

BOOL autoReverseOn;

float multipyThrowPower;

float yAxisGravity;

bool gravityOn;

b2World\* world;

GLESDebugDraw \*m\_debugDraw;

ContactListener\* contactListener;

// 当前被抛投的熊猫角色

Panda\* currentBodyNode;

Panda\* panda1;

// 等待被抛投的熊猫角色

Panda\* panda2;

Panda\* panda3;

Panda\* panda4;

Panda\* panda5;

unsigned char pandasToTossThisLevel;

unsigned char pandaBeingThrown;

//背景

CCSprite\* backgroundLayerClouds;

CCSprite\* backgroundLayerHills;

//粒子效果

CCParticleSystem \*system;

//初始位置

CGPoint cloudLayerStartPosition;

CGPoint hillsLayerStartPosition;

CGPoint particleSystemStartPosition;

CGPoint labelStartingPoint;

//弹弓

CCSprite \*slingShotFront;

CCSprite \*strapFront;

CCSprite \*strapBack;

CCSprite \*strapEmpty;

//BOX2D 物体的起始位置

CGPoint groundPlaneStartPosition;

CGPoint platformStartPosition;

CGPoint pandaStartPosition1;

CGPoint pandaStartPosition2;

CGPoint pandaStartPosition3;

CGPoint pandaStartPosition4;

CGPoint pandaStartPosition5;

// 用于移动场景以查看目标

float worldMaxHorizontalShift;

float previousTouchLocationX;

float adjustY;

float maxScaleDownValue;

float scaleAmount;

float speed;

//标签相关的变量

CCLabelTTF \*highScoreLabel;

CCLabelTTF \*currentScoreLabel;

CGPoint currentScoreLabelStartPosition;

CGPoint highScoreLabelStartPosition;

CGPoint menuStartPosition;

CGPoint slingShotCenterPosition;

CGPoint positionInSling;

//熊猫在空中的白色点痕迹

int dotCount;

int throwCount;

int dotTotalOnEvenNumberedTurn;

int dotTotalOnOddNumberedTurn;

//菜单按钮

CCMenu \*MenuButton;

//拉伸弹弓到最大范围的次数

int stretchBeyondRange;

}

+(MainScene\*) sharedScene;

+(CCScene \*) scene;

-(void) enableDebugMode ;

-(void) setUpParticleSystemSun ;

//碰撞检测机制将会调用以下方法

-(void) proceedToNextTurn:(Panda\*)thePanda;

-(void) stopDotting;

-(void) showPandaImpactingStack:(Panda\*)thePanda;

-(void) showPandaOnGround:(Panda\*)thePanda;

//

-(void) removePreviousDots;

-(void) switchAllStackObjectsToStatic ;

-(void) showBoardMessage:(NSString\*)theMessage;

-(void) showLevelResult;

//场景的移动

-(void) moveScreen:(int)amountToShiftScreen;

-(void) putEverythingInStartingViewOfSlingShot;

-(void) putEverythingInViewOfTargets;

-(void) startScreenPanToTargets;

-(void) startScreenPanToTargetsWithAutoReverseOn;

-(void) autoScreenPanToTargets:(ccTime)delta;

-(void) startScreenPanToSling;

-(void) autoScreenPanToSling:(ccTime)delta;

-(void) moveScreenUp:(ccTime) delta;

-(void) moveScreenDown:(ccTime) delta;

-(void) cancelAutoPan;

//熊猫在弹弓时的调整

-(void) adjustBackStrap:(float)angle;

-(int) returnAmountToShiftScreen:(int)diff;

- (GLfloat) calculateAngle:(GLfloat)x1 :(GLfloat)y1 :(GLfloat)x2 :(GLfloat)y2;

-(BOOL) checkCircleCollision:(CGPoint)center1 :(float)radius1 :(CGPoint) center2 :(float) radius2;

-(void) jumpToLevelResultScene;

//游戏分值

-(void) updatePointsLabel;

-(void) showPoints:(int)pointValue positionToShowScore:(CGPoint)positionToShowScore theSimpleScore:(int)theSimpleScore ;

-(void)showPointsWithImagesForValue:(int) pointValue positionToShowScore:(CGPoint)positionToShowScore ;

@end

从以上代码中可以看到，MainScene类继承自CCLayer，显然将作为游戏的场景出现。在文件的顶部导入了多个类的头文件，其中除了ContactListener.h（见11.3节）外，其他类的作用在之前的内容中都已做了详细的说明。接下来在类声明部分定义了多个变量，这里不再一一解释。最后声明了多个实例方法，其中主要涉及碰撞检测机制要调用的方法，和场景的平移相关的多个方法，和熊猫在弹弓中的调整相关的多个方法，场景过渡的几个方法，以及和游戏分值的更新与显示相关的几个方法。

总的来说，主场景类MainScene包括三大部分的内容：

* 第一部分：实现场景的视觉呈现，包括如何在场景中创建和放置熊猫、平台、屏障物体、外星怪物、背景图等。此外还包括场景的平移等。
* 第二部分：实现与用户的交互机制，即实现对触摸事件的识别和处理。
* 第三部分：游戏的逻辑机制，其中主要涉及碰撞检测机制要调用的方法、游戏分值的更新与实时显示，以及关卡结束后的场景过渡。

除了这三个主要部分之外，还有一些辅助性方法。接下来分成三小节讲解游戏主场景的实现，包括如何实现场景的视觉呈现、如何实现与用户的交互，以及如何实现游戏的逻辑机制。

1. 场景的视觉呈现

在Xcode中切换到MainScene.mm，首先看如何在场景中创建和放置物体。有经验的开发者都知道，Cocos2D游戏场景的初始化都是在以init开头的方法中实现的。

找到init方法，并查看其中的代码如代码清单12-12所示。

1. init方法在场景中创建和放置物体

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[super init])) {

layerInstance = self;

//在场景中启用对触摸事件的支持

self.isTouchEnabled = YES;

//读取GameData中的数据，并设置一些基本的参数

[self readData];

//设置初始位置

[self setInitialPosition];

//添加得分和历史最高得分标签

[self addScoreLabel];

[self updatePointsLabel];

//添加暂停按钮

[self addPauseButton];

//设置box2d

[self initPhysics];

[self initContactListener];

//添加游戏世界中的各种物体

[self addGroundPlaneBody];

[self addFirePlatform];

[self addPanda];

[self addTargets];

//实时更新游戏中的物体所在位置

[self schedule: @selector(tick:)];

//添加进入关卡时的文本介绍

[self addLevelIntroLabel];

//预加载游戏音效和背景音乐

[self initSounds];

}

return self;

}

以上代码实现了以下内容：

1. 首先将当前层赋予一个层的实例变量，以便在后续使用，代码如下：

layerInstance = self;

1. 在场景中启用对触摸事件的支持，代码如下：

self.isTouchEnabled = YES;

1. 读取GameData中所保寸的数据，并设置一些基本的参数，代码如下：

[self readData];

具体的内容可以查看readData方法的实现代码，开发者很容易就可以看懂。

1. 设置屏幕中视觉元素的初始位置，代码如下：

[self setInitialPosition];

具体的内容可以查看setInitialPosition方法的实现代码。

1. 添加当前关卡得分和历史最高得分标签，代码如下：

[self addScoreLabel];

[self updatePointsLabel];

updatePointsLabel方法比较简单，具体的内容可以查看其实现代码。

在addScoreLabel方法的实现代码中，首先使用CCLabelTTF创建并初始化了两个文本标签，分别用于显示当前关卡的实时得分和历史最高得分。注意这里使用了setColor方法设置标签文字的色彩值，相关代码如下：

[currentScoreLabel setColor:ccc3(255,255,255)];

这里使用R、G、B三个色彩值设置标签文字为白色。

1. 添加暂停按钮，代码如下：

[self addPauseButton];

addPauseButton的实现代码很简单，首先使用CCMenuItemImage的类方法创建并初始化了一个图片菜单项，然后创建了一个菜单，并将刚才的图片菜单项添加到菜单中，设置菜单的位置，最后将菜单添加为游戏场景的子节点。

1. 设置Box2D物理世界，代码如下：

[self initPhysics];

[self initContactListener];

initPhysics方法的实现代码如代码清单12-13所示。

1. 设置Box2D物理世界

-(void)initPhysics{

// 设置重力参数

yAxisGravity = -10;

b2Vec2 gravity;

gravity.Set(0.0f, yAxisGravity);

world = new b2World(gravity);

// Do we want to let bodies sleep?

world->SetAllowSleeping(true);

world->SetContinuousPhysics(true);

//[self enableDebugMode];

// 定义地面物体

b2BodyDef groundBodyDef;

groundBodyDef.position.Set(0, 0); // bottom-left corner

// Call the body factory which allocates memory for the ground body

// from a pool and creates the ground box shape (also from a pool).

// The body is also added to the world.调用物体工厂来创建地面物体

b2Body\* groundBody = world->CreateBody(&groundBodyDef);

// Define the ground box shape.创建地面盒定义

b2EdgeShape groundBox;

int worldMaxWidth = screenWidth \* 4; //Box2D世界的最大宽度

int worldMaxHeight = screenHeight \* 3; //Box2D世界的最大高度

// 底部

groundBox.Set(b2Vec2(-4,0), b2Vec2( worldMaxWidth /PTM\_RATIO,0));

groundBody->CreateFixture(&groundBox,0);

// 顶部

groundBox.Set(b2Vec2(-4,worldMaxHeight/PTM\_RATIO), b2Vec2( worldMaxWidth /PTM\_RATIO, worldMaxHeight /PTM\_RATIO));

groundBody->CreateFixture(&groundBox,0);

// 左侧

groundBox.Set(b2Vec2(-4,worldMaxHeight/PTM\_RATIO), b2Vec2(-4,0));

groundBody->CreateFixture(&groundBox,0);

// 右侧

groundBox.Set(b2Vec2( worldMaxWidth /PTM\_RATIO,worldMaxHeight/PTM\_RATIO), b2Vec2(worldMaxWidth /PTM\_RATIO,0));

groundBody->CreateFixture(&groundBox,0);

}

在第9章曾经学习过如何初始化Box2D物理世界。实际上，使用Box2D模板创建的项目中通常都会提供默认的相关代码。在以上的代码中，首先定义y轴的重力参数为10（类似于真实世界的重力），并在创建新的物理世界时指定该数值为重力参数。接下来使用SetAllowSleeping来设置物体是否休眠。如果是，则在没有发生碰撞时让物体进入休眠状态，以节省对资源的占用。接下来设置SetContinuousPhysics为true，从而支持连续物理碰撞。如果需要进入debug测试模式，可以将[self enableDebugMode];这行代码的注释掉。这样，一个新的Box2D物理世界就初始化完成了。

接下来就是创建地面物体，其过程和之前创建一般的物体没有太大区别，这里就不再重复。

initContactListener方法的实现代码中，主要创建一个新的碰撞检测器，并将当前物理世界的碰撞检测机制设置为所创建的碰撞检测器。关于碰撞检测机制的具体实现将在12.3节中详细说明。

1. 添加世界中的各种物体相关代码如下：

[self addGroundPlaneBody];

[self addFirePlatform];

[self addPanda];

[self addTargets];

其中在addGroundPlaneBody方法中，使用GroundPlane的类方法groundWithWorld创建并初始化了一个地平面物体，并将其添加为场景的子节点。

在addFirePlatform方法中，则使用FirePlatform的类方法platformWithWorld创建并初始化了一个弹弓放置平台物体，并将其添加为场景的子节点。

addPanda方法用于在场景中添加要抛投的熊猫。在其实现代码中，首先从GameData中获取当前关卡要抛投的熊猫数量。然后使用Panda类的类方法pandaWithWorld创建并初始化一个熊猫物体，并设置其初始状态。接下来根据要抛投的熊猫数量添加更多的熊猫，并设置熊猫的初始状态。

addTargets方法用于在场景中添加屏障物体和外星怪物。在其实现代码中，首先使用ShootingTargets的类方法setupStackWithWorld创建并初始化屏障物体和外星怪物，并添加为场景层的子节点。接下来给屏障物和怪物一定的时间来掉落，然后将其中的每一部分设置为静态（锁定其位置，直到第一次弹弓发射开始），其相关代码如下：

[self performSelector:@selector(switchAllStackObjectsToStatic) withObject:nil afterDelay:1.0f];

1. 实时更新游戏中物体所在的位置，其代码如下：

[self schedule: @selector(tick:)];

tick方法的实现如代码清单12-14所示。

1. tick方法的实现

-(void) tick: (ccTime) dt

{

//It is recommended that a fixed time step is used with Box2D for stability

//of the simulation, however, we are using a variable time step here.

//You need to make an informed choice, the following URL is useful

//<http://gafferongames.com/game-physics/fix-your-timestep/>

int32 velocityIterations = 8;

int32 positionIterations = 1;

// Instruct the world to perform a single step of simulation. It is

// generally best to keep the time step and iterations fixed.

world->Step(dt, velocityIterations, positionIterations);

//Iterate over the bodies in the physics world

for (b2Body\* b = world->GetBodyList(); b; b = b->GetNext())

{

if (b->GetUserData() != NULL) {

//让Box2D世界的物体和Cocos2D的精灵位置一一对应

CCSprite \*myActor = (CCSprite\*)b->GetUserData();

myActor.position = CGPointMake( b->GetPosition().x \* PTM\_RATIO, b->GetPosition().y \* PTM\_RATIO);

myActor.rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(b->GetAngle());

}

}

}

在以上的代码中，首先对世界对象调用了Step方法，让其开始执行物理世界模拟。然后需要让精灵对象匹配物理模拟的物体。首先遍历了世界对象中的所有物体，并查看物体的userData属性是否为空。如果不为空，则将该对象强制转换为精灵对象，同时更新精灵的位置和角度以匹配物理模拟。

1. 添加进入关卡时的文本介绍，其代码如下：

[self addLevelIntroLabel];

addLevelIntroLabel方法用于调用showBoardMessage方法显示当前关卡的欢迎标签。而showBoardMessage方法则是根据传入的字符串参数显示一个文本标签，并让该标签执行一个序列动作，最后调用removeBoardMessage方法将该标签删除。

进入关卡时显示当前关卡编号效果如图12-5所示。



1. 进入关卡时显示当前关卡编号
2. 预加载游戏音效和背景音乐，其代码如下：

[self initSounds];

在initSounds方法的实现代码中，使用GameSounds类的单例来加载游戏场景所需的音效和背景音乐。

1. 在init场景初始化方法添加了场景中的物体和精灵后，使用了若干个方法以实现游戏中的场景随镜头的移动，重要方法如下：

* startScreenPanToTargets：启动场景向目标的平移。
* startScreenPanToTargetsWithAutoReverseOn：启动场景向目标的平移，并启用自动转向。
* autoScreenPanToTargets：让屏幕场景自动向目标平移。
* startScreenPanToSlingIfScoringIsNotOccuring：在没有得分时，让屏幕场景向弹弓所在位置平移。
* startScreenPanToSling：启动屏幕场景向弹弓的平移。

限于篇幅，这里不列出所有和场景平移相关的辅助方法，开发者可以查看示例项目中的相关代码。

1. 如何与用户交互

在实现了场景的视觉呈现和平移后，将说明在场景中如何实现与用户交互。在Angry Panda游戏中，与玩家的交互主要是通过触摸屏幕来实现。

具体来说，存在两种交互，一种是玩家用手指拉动弹弓发射熊猫，而另一种则是玩家来回移动屏幕。后一种交互相对比较简单，而拉动弹弓发射熊猫则是这款游戏的核心精髓所在。

在第6章曾经详细了解过在Cocos2D中触摸事件的处理机制。在Cocos2D中处理触摸事件可以采用两种不同的代理对象处理协议，分别是标准代理对象处理协议和目标代理对象处理协议。这里采用前一种处理协议，即和UIKit中默认的触摸事件处理机制相同的方式。

#### ccTouchesBegan方法

ccTouchesBegan方法的实现如代码清单12-15所示。

1. ccTouchesBegan方法的实现

-(void) ccTouchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event {

for( UITouch \*touch in touches ) {

CGPoint location = [touch locationInView: [touch view]];

location = [[CCDirector sharedDirector] convertToGL: location];

previousTouchLocationX = location.x;

if ( throwInProgress == NO ) {

currentBodyNode.body->SetType(b2\_staticBody);

}

}

}

ccTouchesBegan主要实现当触摸事件开始时的处理机制。在以上代码中，首先使用for循环遍历了所有的触摸点。使用CGPoint变量保存触摸点在视图中的位置，并使用CCDirector的单例将其转换为OpenGL视图中的位置，并将x坐标保存到一个变量中。此时如果熊猫不在抛投中，则将当前物体设置为静态物体。

#### ccTouchesMoved方法

ccTouchesMoved方法的实现如代码清单12-16所示。

1. ccTouchesMoved方法的实现

-(void)ccTouchesMoved:(NSSet \*)touches withEvent:(UIEvent \*)event {

if ( autoPanningInProgress == YES) {

[self cancelAutoPan];

} //1

//触摸交互

for( UITouch \*touch in touches ) {

CGPoint location = [touch locationInView: [touch view]];

location = [[CCDirector sharedDirector] convertToGL: location]; //2

//移动弹弓中的熊猫

//如果场景在起始位置(self.position.x == 0) ，且还未开始抛投，而玩家手指正在弹弓周围触摸

if ( ( [self checkCircleCollision:location :2 :slingShotCenterPosition:maxStretchOfSlingShot] == YES || slingShotPandaInHand == YES ) && throwInProgress == NO && areWeInTheStartingPosition == YES ) //3

{

if ( slingShotPandaInHand == NO) {

positionInSling = slingShotCenterPosition;

slingShotPandaInHand = YES;

strapBack.visible = YES;

strapFront.visible = YES;

strapEmpty.visible = NO;

} //4

int currentAngle = currentBodyNode.body->GetAngle() ;

b2Vec2 bodyPos = currentBodyNode.body->GetWorldCenter(); //5

for( UITouch \*touch in touches ) {

CGPoint location = [touch locationInView: [touch view]];

location = [[CCDirector sharedDirector] convertToGL: location]; //6

float radius = maxStretchOfSlingShot; //radius of slingShot

GLfloat angle = [self calculateAngle:location.x :location.y :slingShotCenterPosition.x :slingShotCenterPosition.y] ; //7

//如果玩家在弹弓的最大拉伸半径范围内移动熊猫

if ( [self checkCircleCollision:location :2 :slingShotCenterPosition: radius] == YES) {

positionInSling = ccp(location.x , location.y); //8

//根据触摸点和弹弓中心点的位置来判断弹弓皮带的缩放大小

float scaleStrap = (abs( slingShotCenterPosition.x - location.x )) / radius;

scaleStrap = scaleStrap + 0.3; //9

if ( scaleStrap > 1) {

scaleStrap = 1;

} //10

strapFront.scaleX = scaleStrap;

strapBack.scaleX = strapFront.scaleX ; //9

} else {

//如果玩家向弹弓的最大拉伸半径范围外移动熊猫

GLfloat angleRadians = CC\_DEGREES\_TO\_RADIANS (angle - 90);

positionInSling = ccp( slingShotCenterPosition.x - (cos(angleRadians) \* radius) , slingShotCenterPosition.y + (sin(angleRadians) \* radius) );

strapFront.scaleX = 1;

strapBack.scaleX = 1;

if(stretchBeyondRange ==0){

[[GameSounds sharedGameSounds] playStretchSlingshotSound];

stretchBeyondRange++;

}

} //10

strapFront.rotation = angle - 90;

[self adjustBackStrap:(float)angle ]; //11

//设置熊猫物体的位置

b2Vec2 locationInMeters = b2Vec2(positionInSling.x / PTM\_RATIO, positionInSling.y / PTM\_RATIO);

currentBodyNode.body->SetTransform( locationInMeters , CC\_DEGREES\_TO\_RADIANS( currentAngle)); //12

}

[currentBodyNode spriteInPulledBackSlingState]; //13

}

//如果不满足以上条件则移动屏幕中的场景

else {

int amountToShiftScreen;

int diff = (location.x - previousTouchLocationX);

amountToShiftScreen = [self returnAmountToShiftScreen:diff];

// 如果玩家手中没有熊猫，则将屏幕场景前后平移

if ( self.position.x <= 0 && self.position.x >= worldMaxHorizontalShift && slingShotPandaInHand == NO ) {

areWeInTheStartingPosition = NO;

[self moveScreen:(int)amountToShiftScreen]; //14

if (self.position.x > 0 ) { // 如果场景向左移动的过多，则重新设置到起始位置

[self putEverythingInStartingViewOfSlingShot]; //15

}

else if (self.position.x < worldMaxHorizontalShift ) //16

{

// 如果场景向相反方向移动的过多，则重新设置到最大位置

[self putEverythingInViewOfTargets]; //17

}

}

}

previousTouchLocationX = location.x; //18

}

}

以上方法的实现代码比较复杂，同时也是互动中最核心的部分之一，因此这里将按照代码注释的数字顺序一一进行解释：

1. 如果场景正在自动平移，则调用cancelAutoPan方法取消自动平移。
2. 使用for循环遍历触摸点，获取并保存触摸点的位置，并转换为Cocos2D世界中的位置。
3. 使用if判断，如果场景正处于初始位置，且还没有开始抛投熊猫，且玩家正在弹弓的周围触摸。
4. 设置熊猫状态为正在弹弓中，将弹弓皮带等精灵对象的可视属性开启。
5. 获取当前物体的旋转角度，并获取所在世界的中心
6. 再次使用for循环遍历触摸点，获取并保存触摸点的位置，并转换为Cocos2D世界中的位置
7. 计算弹弓当前位置到中心位置的角度和拉伸半径
8. 判断玩家是否在弹弓的最大拉伸半径范围内移动熊猫，如果是则获取当前所在位置
9. 根据触摸点和弹弓中心点的位置来计算弹弓皮带的缩放大小
10. 如果玩家向弹弓的最大拉伸半径范围外移动熊猫，则调整弹弓皮带的缩放大小为原始大小，同时使用GameSounds的单例对象播放一个短音效。
11. 设置弹弓皮带的旋转，并调用addJustBackStrap方法进行相应的调整。
12. 设置熊猫物体的位置
13. 设置熊猫的视觉呈现状态。
14. 如果不满足3）中的条件，则将屏幕场景前后平移
15. 如果场景向左移动的过多，则重新设置到初始位置。
16. 如果场景向相反方向移动的过多，则重新设置到最大位置。
17. 将所有场景元素均放置到目标视角。
18. 将当前位置的x坐标保存到变量中。

以上代码中还调用了一些辅助方法，有兴趣的开发者可以查看示例项目中的相关代码。

#### ccTouchesEnded方法

最后看触摸事件结束后的处理机制，ccTouchesEnded方法的实现如代码清单12-17所示。

1. ccTouchesEnded方法的实现

- (void)ccTouchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event

{

//在触碰位置添加新的物体和精灵

for( UITouch \*touch in touches ) {

CGPoint location = [touch locationInView: [touch view]];

location = [[CCDirector sharedDirector] convertToGL: location]; //1

if ( slingShotPandaInHand == YES) {

[[GameSounds sharedGameSounds] playReleaseSlingshotSound];

[self switchAllStackObjectsToDynamic]; //2

throwCount ++;

dotCount = 0;

throwInProgress = YES; //3

currentBodyNode.body->SetType(b2\_dynamicBody);

currentBodyNode.body->SetAwake(true); //4

strapBack.visible = NO;

strapFront.visible = NO;

strapEmpty.visible = YES; //5

// 计算速度

speed = (abs( slingShotCenterPosition.x - positionInSling.x )) + (abs( slingShotCenterPosition.y - positionInSling.y )) ;

speed = speed / 5;

speed = speed \* multipyThrowPower; //6

// targetPosition即触摸点

b2Vec2 targetInWorld = b2Vec2(location.x, location.y); //7

// 判断熊猫的实际运动方向，从弹弓中心到触摸点

b2Vec2 direction = b2Vec2(slingShotCenterPosition.x - location.x, slingShotCenterPosition.y - location.y);

direction.Normalize(); //8

// 移动物体， 设置线性速度

currentBodyNode.body->SetLinearVelocity( speed \* direction ); //9

slingShotPandaInHand = NO;

[currentBodyNode spriteInAirState]; //10

// 添加熊猫的运行轨迹白点

[self removePreviousDots]; //11

dottingOn = YES; //正在添加白点中

[self schedule:@selector(placeWhiteDots:) interval:1.0f/45.0f]; //12

//确保在6秒后将throwInProgress设置为NO

[self unschedule:@selector(timerAfterThrow:)];

[self schedule:@selector(timerAfterThrow:) interval:6.0f]; //13

if (location.x < slingShotCenterPosition.x ) {

[self startScreenPanToTargetsWithAutoReverseOn];

}

} //14

else if (continuePanningScreenOnFingerRelease == YES) {

if ( panningTowardSling == YES) {

[self startScreenPanToSling];

} else {

[self startScreenPanToTargets];

}

} //15

}

stretchBeyondRange =0; //16

}

以上方法的实现代码比较复杂，同时也是互动中最核心的部分之一，因此这里将按照注释的数字顺序一一进行解释：

1. 使用for循环遍历触摸点，获取并保存触摸点的位置，并转换为Cocos2D世界中的位置。
2. 使用if判断，如果触摸结束时熊猫在手中，则播放释放弹弓的音效，同时将所有的屏蔽物体和怪物转换为动态物体。
3. 将已抛投的熊猫数量加1，将白点数量设置为0，将熊猫在抛投中这个布尔变量设置为YES。
4. 将当前物体（熊猫）设置为动态物体，并设置其为苏醒状态。
5. 设置弹弓皮带的显示属性为隐藏。
6. 计算熊猫的移动速度。
7. 使用b2Vec2变量保存触摸点的位置信息。
8. 判断熊猫从弹弓中心到触摸点的实际运动方向，并将其正交化。
9. 根据所计算出的移动速度和实际运动方向设置熊猫的线性速度。
10. 设置熊猫的视觉显示状态。
11. 删除之前的运动轨迹（白点）。
12. 使用一定的间隔放置白点，从而添加熊猫在空中的运动轨迹。
13. 在6秒将熊猫在抛投中的状态设置为NO。
14. 如果熊猫的x坐标小于弹弓中心点的x坐标，则调用方法让场景回复到初始状态。
15. 根据参数值的不同选择不同的场景平移方法。
16. 设置弹弓拉伸超出范围的变量为0。
17. 实现游戏的逻辑机制

Angry Panda示例游戏的逻辑机制主要包括和碰撞检测机制相关的方法、游戏分值的更新与实时显示，以及关卡结束后的场景过渡等，下面将对相关的方法进行简单说明：

#### 和碰撞检测机制相关的方法

* proceedToNextTurn：当熊猫碰到地面时切换到下一次抛投。在该方法的实现代码中，首先取消对timerAfterThrow的消息预定，然后在1秒后预定moveNextPandaIntoSling消息。
* showPandaImpactingStack：当熊猫碰到屏蔽物体时显示相应的动画。
* showPandaOnGround：当熊猫碰到地面时显示相关的状态。首先判断熊猫是否是和地面发生碰撞的同一个熊猫，如果是，则调用makePandaStaticOnGround方法让熊猫变成静态物体，然后使用performSelector方法以调用fadeThenRemove方法，在几秒钟后让熊猫淡出并消失。

#### 游戏分值的更新与实时显示

* updatePointsLabel：更新游戏得分标签。
* somethingJustScored：辅助方法，当有得分时调用，在该方法的实现代码中，首先设置布尔变量值为YES，然后取消对resetSomethingJustScored的消息预定，最后在3秒后预定resetSomethingJustScored消息。
* resetSomethingJustScored：重置得分后的属性。
* showPoints：用于显示游戏中的得分。
* showPointsWithImagesForValue：使用分值图片来显示得分。

#### 关卡结束时的场景过渡

* doPointBonusForExtraPandas：为游戏中剩下的熊猫获得额外奖励得分。
* showLevelResult：在关卡结束时进入显示当前关卡的游戏结果界面。其实现如代码清单12-18所示。

1. showLevelResult方法的实现

-(void) showLevelResult {

if ( pointTotalThisRound >= pointsToPassLevel ) //1

{

[self doPointBonusForExtraPandas]; //2

GameData \*data = [GameData sharedData];

data.currentLevelScore = pointTotalThisRound;

[[GameData sharedData] sumOfScoresForAllLevels:pointTotalThisRound];

[[GameData sharedData] setHighScoreForCurrentLevel:pointTotalThisRound]; //设置最高得分

[[GameSounds sharedGameSounds]levelClear];

[GameData sharedData].currentLevelSolved = YES;

[[GameData sharedData]levelComplete]; //3

} else {

[[GameData sharedData] setHighScoreForCurrentLevel:pointTotalThisRound]; //即便未能通过关卡也会设置最高得分

GameData \*data = [GameData sharedData];

data.currentLevelScore = pointTotalThisRound;

[GameData sharedData].currentLevelSolved = NO;

[[GameSounds sharedGameSounds]levelLose]; //4

}

[self performSelector:@selector(jumpToLevelResultScene) withObject:nil afterDelay:3.0f]; //5

}

下面根据代码注释行的编号进行解释：

1. 判断当前关卡的得分是否大于等于要通关的必需分数。
2. 如果是，则首先为多出来的熊猫提供奖励分数。
3. 将当前关卡的相关数据保存到GameData中，并使用GameSounds的单例对象来播放音效。
4. 如果当前关卡得分小于要通关的必需分数，仍然需要将当前关卡的相关数据保存到GameData中，并使用GameSounds的单例对象来播放音效。
5. 延迟3秒后调用jumpToLevelResultScene方法以切换到游戏关卡结束的场景(LevelResult)。

除了这三大类方法外，MainScene类中还有一些辅助性方法，这里就不再一一详述。

下一节将实现游戏中的碰撞检测机制，从而最终完成游戏的核心部分。

## 游戏中的碰撞检测机制

在完成了MainScene类之后，Angry Panda游戏已经基本成型，但为了让Box2D世界真正和游戏融合在一起，还需要实现最后一部分，即游戏中的碰撞检测机制。

1. 创建ContactListener类

在Xcode中右键单击AngryPanda组，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为ContactListener.mm，单击Save。

在Xcode中切换到ContactListener.h，并替换其中的代码，如代码清单12-19所示。

1. ContactListener.h文件

#import "Box2D.h"

#import "GameSounds.h"

class ContactListener : public b2ContactListener

{

private:

void BeginContact(b2Contact\* contact);

void PreSolve(b2Contact\* contact, const b2Manifold\* oldManifold);

void PostSolve(b2Contact\* contact, const b2ContactImpulse\* impulse);

void EndContact(b2Contact\* contact);

};

以上代码比较简单，定义ContactListener类，其继承自b2ContactListener。

提示 本书第9章详细讲解过b2ContactListener，其主要作用是负责监听和处理Box2D世界中发生的物体碰撞。

打开示例项目中的ContactListener.mm，并将其中的代码粘贴到你所创建的项目文件中。限于篇幅，这里就不再列出所有的实现代码。

1. 碰撞检测机制的实现

碰撞检测和处理机制的核心部分在ContactListener::BeginContact中。

在Angry Panda这款示例游戏中，碰撞检测机制的逻辑判断过程分为6种情况：

1. 碰撞的两个物体中有一个是熊猫，另一个是地平面，即熊猫碰到地平面上。

判断方式：使用isKindOfClass方法。此时将停止绘制飞行轨迹，显示熊猫在地面上的状态，并前进到下一回合。

1. 碰撞的两个物体中有一个是熊猫，而另一个是遮挡物，即熊猫碰到遮挡物上。

判断方式：使用isKindOfClass方法进行判断。此时会播放一个音效，停止绘制在空中的轨迹，并调用熊猫撞击遮挡物时触发的方法，播放分解动画。此时如果和熊猫碰撞时会有得分，则显示新的得分。

1. 碰撞的两个物体中有一个是熊猫，而另一个是外星怪物，即熊猫碰到外星怪物上。

判断方式：此时会首先停止控制在空中的轨迹，然后调用熊猫撞击外星怪物的方法。此时如果和熊猫碰撞时有份，则显示新的得分，并让怪物受到伤害或分解。

1. 碰撞的两个物体中有一个是遮挡物，而另一个是外星怪物，即遮挡物碰到外星怪物。

判断方式：此时主要显示新的得分，并让外星怪物受到伤害或分解。

1. 外星怪物碰到地平面上。

判断方式：此时主要显示新的得分，并让外星怪物受到伤害或分解。

1. 遮挡物碰到地平面上。

判断方式：此时主要显示新的得分，并让遮挡物分解。

以上代码虽然看起来复杂，但只要看懂其中的一种情况，其他都完全类似。

这6种情况的方法实现代码类似，以第一种情况为例，相关代码如代码清单12-20所示。

1. 熊猫碰到地平面上的碰撞检测机制

b2Body\* bodyA = contact->GetFixtureA()->GetBody();

b2Body\* bodyB = contact->GetFixtureB()->GetBody(); //1

PhysicsNode\* physicsNodeA = (PhysicsNode\*)bodyA->GetUserData();

PhysicsNode\* physicsNodeB = (PhysicsNode\*)bodyB->GetUserData(); //2

//熊猫碰到地平面上

if ([physicsNodeA isKindOfClass:[Panda class]] && [physicsNodeB isKindOfClass:[GroundPlane class]]) //3

{

Panda\* thePanda = (Panda\*)physicsNodeA;

[[MainScene sharedScene] stopDotting];

[[MainScene sharedScene] showPandaOnGround:thePanda ];

[[MainScene sharedScene] proceedToNextTurn:thePanda ]; //4

}

else if ([physicsNodeA isKindOfClass:[GroundPlane class]] && [physicsNodeB isKindOfClass:[Panda class]]) //5

{

Panda\* thePanda = (Panda\*)physicsNodeB;

[[MainScene sharedScene] stopDotting];

[[MainScene sharedScene] showPandaOnGround:thePanda ];

[[MainScene sharedScene] proceedToNextTurn:thePanda]; //6

}

下面按照代码行的注释顺序依次进行解释：

1）定义两个b2Body物体，并从碰撞集合中使用GetFixture()和GetBody()方法获取了发生碰撞的两个物体，并赋予所定义的b2Body物体。

2）定义两个BodyNode物体，并使用GetUserData()方法将物体转换成BodyNode物体并赋予所定义的物体。

3）判断当出现碰撞物体A属于熊猫，碰撞物体B属于地平面物体时，

4）创建一个Panda对象，并将碰撞物体A赋予该对象，使用MainScene的单例停止描绘熊猫在空中的运行轨迹，显示熊猫在地面上的状态，并准备好抛投下一个熊猫。

5）判断当出现碰撞物体A属于地平面物体，碰撞物体B属于熊猫时，

6）创建一个Panda对象，并将碰撞物体B赋予该对象，使用MainScene的单例停止描绘熊猫在空中的运行轨迹，显示熊猫在地面上的状态，并准备好抛投下一个熊猫。

其余5种情况的代码与之类似，这里就不再一一解释。

至此，游戏的碰撞检测机制也已准备就绪。

编译运行项目，Angry Birds游戏就完成了。在模拟器中使用鼠标拉动弹弓如图12-6所示。



1. 在模拟器中使用鼠标拉动弹弓

释放鼠标后，熊猫在空中飞行并留下白色轨迹，如图12-7所示。



1. 熊猫在空中飞行并留下白色轨迹

熊猫碰到屏障物体和外星怪物后，会产生相应的分值，如图12-8所示。



1. 熊猫碰到屏障物体和外星怪物的效果

## 本章小结

本章在第11章的基础上学习使用Box2D物理引擎创建游戏中的各类物体，构建游戏的主场景，实现游戏的交互机制，并使用碰撞检测机制来完成最终的互动。关卡场景中的整个游戏世界都是基于第9章的Box2D物理引擎来构建的。

通过Angry Panda项目的设计与实现充分结合之前所学习的Cocos2D理论知识，带领开发者一步步熟悉如何在实际的项目中使用基础篇所学习到的知识。

从第13章开始，将介绍Cocos2D比较高级的知识。