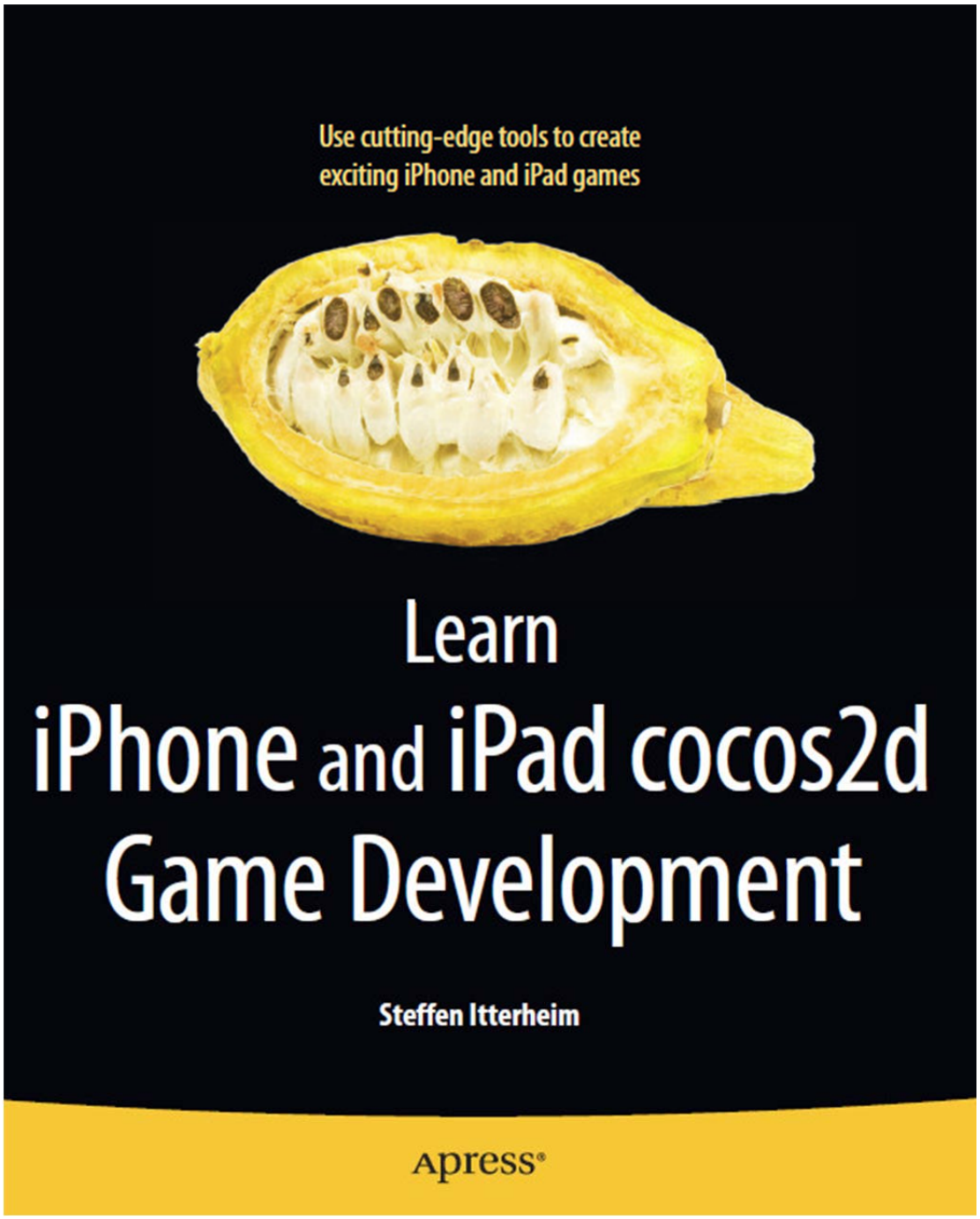

译者：杨栋
邮箱：yangdongmy@gmail.com

第九章
粒子效果

游戏开发者通常使用粒子系统来制作视觉特效。粒子系统会发射大量细小的粒子和非常高效地渲染这些粒子－比渲染精灵要高效的多。你可以模拟下雨，火焰，雪，爆炸，蒸汽拖尾和很多其它视觉效果。

粒子系统由很多属性来驱动。这里的“很多”大概是30种左右的属性，它们不仅影响单个粒子的外观和行为，而且也影响着整体的粒子效果。粒子效果是所有的粒子一起工作所创造出的独有的视觉效果。一个或者十个粒子是做不出火焰效果的，你需要几十个甚至几百个粒子以正确的方式运行才能够创造出火焰效果。

创造令人信服的粒子效果是一个“尝试-错误-再尝试”的过程。在源代码里尝试着改变属性，编译查看得到的粒子效果，然后再改属性和编译查看效果，是个很麻烦和费时的过程。所以我们需要一个粒子效果设计工具。我推荐的是Particle Designer（粒子效果设计器)。我将在本章介绍这个工具。

粒子效果演示实例

cocos2d的源码中自带了很多内置的粒子效果实例，这些例子可以教会你cocos2d能够创造的粒子效果。如果你想在简单修改以后就直接使用这些例子中的特效，你可以直接修改使用CCParticleExamples.m中的代码，或者创建一个继承类。创建例子中的粒子效果和使用普通CCNode对象的方式是一样的，因为它们就是继承自CCNode类的。

我创建了一个ParticleEffects01项目，用于展示cocos2d源码中自带的粒子效果。你可以通过点击屏幕来查看各个例子，也可以用你的手指直接拖动屏幕上的粒子来移动它们。很多粒子效果在移动的时候和静止状态下的表现是很不一样的，比如静止状态下可能看上去是一个白色的团状物，在移动的时候却很适合作为引擎喷出的尾气效果。

只有一种粒子效果不能被移动：如图9-1所示的像CCParticleExplosion（爆炸）这样的一次性效果。爆炸效果的特别之处是：所有的粒子会被一次性射出，然后粒子发射就结束了。其它的粒子效果都会一直发射粒子：在一些粒子因为生命结束而消失的同时，新的粒子还在不断地生成。使用这些粒子效果的挑战是如何保持屏幕上会同时显示的粒子总数。

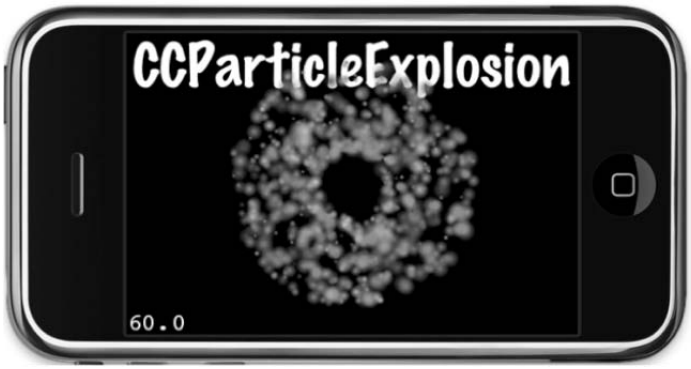

图9-1. cocos2d提供的CCParticleExplosion爆炸展示

列表9-1展示了ParticleEffects01项目中使用的相关方法。通过在switch判断中使用particleType这个变量，会生成相应的内置粒子效果。你可能注意到我使用了一个CCParticleSystem指针来存放所有的粒子，因此我只需要在runEffect方法的最后运行一次addChild就够了。我在这里生成的每个粒子效果都是继承自CCParticleSystem。

列表9-1. 使用内置的粒子效果

```
-(void) runEffect
{
	// 清除任何以前使用过的粒子效果
	[self removeChildByTag:1 cleanup:YES];

	CCParticleSystem* system;

	switch (particleType)
	{
		case ParticleTypeExplosion:
			system = [CCParticleExplosion node];
			break;
		case ParticleTypeFire:
			system = [CCParticleFire node];
			break;
		case ParticleTypeFireworks:
			system = [CCParticleFireworks node];
			break;
		case ParticleTypeFlower:
			system = [CCParticleFlower node];
			break;
		case ParticleTypeGalaxy:
			system = [CCParticleGalaxy node];
			break;
		case ParticleTypeMeteor:
			system = [CCParticleMeteor node];
			break;
```

```
                case ParticleTypeRain:
                        system = [CCParticleRain node];
                        break;
                case ParticleTypeSmoke:
                        system = [CCParticleSmoke node];
                        break;
                case ParticleTypeSnow:
                        system = [CCParticleSnow node];
                        break;
                case ParticleTypeSpiral:
                        system = [CCParticleSpiral node];
                        break;
                case ParticleTypeSun:
                        system = [CCParticleSun node];
                        break;
                default:
                        // 这里不运行任何代码
                        break;
        }

        [self addChild: system z:1 tag:1];
        [label setString:NSStringFromClass([system class])];
}

-(void) setNextParticleType
{
        particleType++;
        if (particleType == ParticleTypes_MAX)
        {
                particleType = 0;
        }
}
```

注：在上述例子中，NSStringFromClass方法很有用。它可以让你避免手动输入十几个字母长的粒子效果名称，是Objective-C提供的一个很好用的功能。如果你要在C++中做同样事情的话，你可能就要痛苦地咬自己的脚趾头了。

对于任何的游戏代码来说，NSStringFromClass和相关的方法很少会被用到，但它们是很好的调试和日志工具。你可以在苹果的Apple’s Foundation Function Reference文档中找到相关信息。：
http://developer.apple.com/mac/library/documentation/Cocoa/Reference/Foundation/Miscellaneous/Foundation_Functions/Reference/reference.html

如果你直接在自己的项目中使用上述范例中的粒子效果，你可能会看到图9-2中

难看的方形像素点。产生这些方形像素点的原因是cocos2d内置的粒子效果需要加载一个指定的贴图：fire.png。这张图片可以在cocos2d源码中的Resources/Images文件夹中找到。在没有这样贴图的情况下，你还是可以生成很漂亮的粒子效果，前提是粒子要很小才行。你可以把fire.png加到你的Xcode项目中，这样就可以看到内置粒子效果的原始效果了。

图9-2. 如果你使用的cocos2d内置效果，像本图的CCParticleFireworks一样，显示的是很大的，方形的粒子，那是因为你忘记把fire.png这张图片添加到Xcode项目中了。

手动生成粒子效果

你可以很容易地创建你自己的CCParticleSystem的子类。不容易的是用这个子类生成令人信服的粒子效果，更不用说让最终的效果达到你一开始设想的样子了。以下是按照功能来分组的粒子效果属性，这些属性决定了粒子系统的外观和行为：

- emitterMode = gravity
  - gravity, centerOfGravity
  - radialAccel, radialAccelVar
  - speed, speedVar
  - tangentialAccel, tangentialAccelVar
- emitterMode = radius
  - startRadius, startRadiusVar, endRadius, endRadiusVar
  - rotatePerSecond, rotatePerSecondVar
- duration
- posVar
- positionType
- startSize, startSizeVar, endSize, endSizeVar
- angle, angleVar
- life, lifeVar
- emissionRate
- startColor, startColorVar, endColor, endColorVar
- blendFunc, blendAdditive
- texture

你可以想像的到，这里有很多可以调整的属性。当然，首先你要了解各个属性都是拿来干什么用的。让我们从头开始：创建一个继承自CCParticleSystem的新类。

实际上，你应该继承自CCPointParticleSystem（点粒子系统）或者CCQuadParticleSystem（方形粒子系统）。点粒子系统在第一代和第二代iOS设备上运行比方形粒子系统要快一点。但是点粒子系统在第三代和第四代设备上，比如iPhone 3GS，iPad和iPhone4，运行的不是很流畅。这是因为这些设备的CPU架构改变了。第三代和第四代设备使用了ARMv7 CPU架构，里面包含了很多优化和新功能，比如增加了一个“矢量浮点处理器”（Vector Floating-Point Processor）和一个SIMD指令集（SIMD Instruction Set）（NEON），而CCQuadParticleSystem则很好的利用了这些新功能。

如果你无法决定要用哪个粒子系统，你就选择CCQuadParticleSystem，因为它在所有设备上都可以流畅运行，由它生成的效果和点粒子系统一样。或者你可以让cocos2d通过构建目标（build target）来替你判断应该使用哪个粒子系统。**列表9-2**展示了我在ParticleEffects02项目中添加的ParticleEffectSelfMade文件的代码。

列表9-2. 让新类继承自最优化的粒子系统

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"

// 基于不同的目标设备，
// ParticleEffectSelfMade 会继承自 CCPointParticleSystem 或者 CCQuadParticleSystem
@interface ParticleEffectSelfMade : ARCH_OPTIMAL_PARTICLE_SYSTEM
{
}
@end
```

我们使用预处理器定义 ARCH_OPTIMAL_PARTICLE_SYSTEM，而不是实际的类名称，在代码编译过程中来判断应该继承自哪个粒子系统。cocos2d中的这个预处理器定义是基于设备处理器架构的，得到的判断结果将会是CCQuadParticleSystem或者CCPointParticleSystem：

```
// 为每一个CPU架构指定最佳的粒子系统
#ifdef __ARM_NEON__
        // armv7
        #define ARCH_OPTIMAL_PARTICLE_SYSTEM CCQuadParticleSystem
#elif __arm__ || TARGET_IPHONE_SIMULATOR
        // armv6 或者模拟器
        #define ARCH_OPTIMAL_PARTICLE_SYSTEM CCPointParticleSystem
#else
        #error(unknown architecture)
#endif
```

现在，让我们来看一下 ParticleEffectSelfMade 类的实现代码，它使用了所有可用的属性。我将解释每一个属性，不过最好的方法是你自己调试修改代码，看到真实的粒子效果。**列表9-3**展示了ParticleEffects02项目中的代码，你也可以在代码中看到简短的注释：

列表9-3. 手动设置粒子系统的属性

```
#import "ParticleEffectSelfMade.h"
@implementation ParticleEffectSelfMade
-(id) init
{
	return [self initWithTotalParticles:250];
}

-(id) initWithTotalParticles:(int)numParticles
{
	if ((self = [super initWithTotalParticles:numParticles]))
	{
		self.duration = kCCParticleDurationInfinity;
		self.emitterMode = kCCParticleModeGravity;

		// 一些属性只针对某个指定的 emitterMode（粒子发射器模式）
		if (self.emitterMode == kCCParticleModeGravity)
		{
			self.gravity = CGPointMake(-50, -90);
			self.centerOfGravity = CGPointMake(-15, 0);
			self.radialAccel = -90;
			self.radialAccelVar = 20;
			self.tangentialAccel = 120;
			self.tangentialAccelVar = 10;
			self.speed = 15;
			self.speedVar = 4;
		}
		else if (self.emitterMode == kCCParticleModeRadius)
		{
			self.startRadius = 100;
			self.startRadiusVar = 0;
			self.endRadius = 10;
			self.endRadiusVar = 0;
			self.rotatePerSecond = -180;
			self.rotatePerSecondVar = 0;
		}
		self.position = CGPointZero;
		self.posVar = CGPointZero;
		self.positionType = kCCPositionTypeFree;
```

```
        self.startSize = 40.0f;
        self.startSizeVar = 0.0f;
        self.endSize = kCCParticleStartSizeEqualToEndSize;
        self.endSizeVar = 0;

        self.angle = 0;
        self.angleVar = 0;

        self.life = 5.0f;
        self.lifeVar = 1.0f;

        self.emissionRate = 30;
        self.totalParticles = 250;

        startColor.r = 1.0f;
        startColor.g = 0.25f;
        startColor.b = 0.12f;
        startColor.a = 1.0f;
        startColorVar.r = 0.0f;
        startColorVar.g = 0.0f;
        startColorVar.b = 0.0f;
        startColorVar.a = 0.0f;
        endColor.r = 0.0f;
        endColor.g = 0.0f;
        endColor.b = 0.0f;
        endColor.a = 1.0f;
        endColorVar.r = 0.0f;
        endColorVar.g = 0.0f;
        endColorVar.b = 1.0f;
        endColorVar.a = 0.0f;

        self.blendFunc = (ccBlendFunc){GL_SRC_ALPHA, GL_DST_ALPHA};

        // 或者使用以下简写方式将混合方式设置为GL_SRC_ALPHA, GL_ONE
        // self.blendAdditive = YES;

        self.texture = [[CCTextureCache sharedTextureCache] addImage:@"fire.png"];
    }
    return self;
}
@end
```

变化度属性

上述代码中，你会注意到很多属性有一个配对的属性后缀是Var。这些是变化度

属性，他们用于决定对应属性允许的模糊度范围。比如，属性 life=5 和 lifeVar = 1。这些数值意味着每个粒子的平均寿命是5秒。这个变化度 lifeVar，允许生命值在 5-1 和 5+1 之间变化。这样每个粒子都会得到一个随机的生命值，介于4秒和6秒之间。

如果你不想设置任何可变度，你可以将Var后缀的变量设置为0。可变度会带给粒子效果动态和变化的行为和外观。不过你在设计新效果时，可变度可能会让你迷惑。因此，除非你已经有了一些设计经验，我建议你开始的时候还是使用很小的可变度值为好，或者根本不用。

粒子数量

让我们通过 totalParticles 属性开始深入了解粒子效果。totalParticles 用于设置粒子效果中的粒子总数。通常我们通过 initWithTotalParticles 方法来设置 totalParticles 变量，你可以在之后改变它的值。粒子的数量对粒子效果的外观和运行效率有直接的影响。

```
-(id) init
{
	return [self initWithTotalParticles:250];
}
```

如果粒子数量太少，你就得不到好看的发光效果（假设我们要做个发光的粒子效果），不过用于生成一个围绕着玩家角色转的一圈星星可能是够了 - 当游戏中的角色撞上墙的时候。如果使用了过多的粒子，很多粒子就会在别的粒子上渲染，导致粒子相互叠加，最终的效果就是个大的白团。此外，使用太多粒子会导致游戏运行帧率大大降低。这也是为什么 Particle Designer 这个软件不允许创建拥有超过2000个粒子以上的效果。

注：一般而言，你的目标是应用最少的粒子创造所需的效果。单个粒子的大小对游戏运行效率也有很大的影响 - 单个粒子越小，运行效率越高。

发射器(Emitter)持续时间

持续时间（duration）属性决定着发射器发射粒子的时间长短。如果设为2，发射器将会在两秒钟的时间里持续生成粒子，然后就停止了。代码非常简单：

```
self.duration = 2.0f;
```

如果你想在粒子系统停止发射粒子和粒子都消失以后，将粒子系统的节点从它的父节点删除的话，你可以将 autoRemoveOnFinish 属性设置为 YES:

```
self.autoRemoveOnFinish = YES;
```

autoRemoveOnFinish属性是个便利功能，它只有和粒子系统结合使用时才会有意义（这里的粒子系统必须是非永久运行的，也就是粒子发射器会停止发射粒子，所有已存在的粒子也最终会消失）。

cocos2d为永久运行的粒子效果定义了一个常量：
kCCParticleDurationInfinity（等于：-1）。大多数粒子效果属于这个类别：

```
self.duration = kCCParticleDurationInfinity;
```

发射器模式

cocos2d的粒子系统存在两种发射器模式：重力（Gravity）和半径（Radius）。它们由emitterMode属性来控制。虽然这两个模式的大多数参数相同，但是生成的粒子效果却完全不同。你可以通过比较图9-3和图9-4看到它们所生成效果的不同之处。这两个模式都有几个独有的属性，如果你在一个模式里应用了不属于此模式的属性，你将会得到与以下出错信息类似的异常抛出：

```
ParticleEffects[6332:207] *** Terminating app due to uncaught exception
                'NSInternalInconsistencyException', reason: 'Particle Mode should be Radius'
```

发射器模式：重力（Gravity）

重力模式可以让粒子向一个中心点飞去，或者飞离一个中心点。它的优点是可以创造出非常动态和自然的效果。以下代码用于设定重力模式：

```
self.emitterMode = kCCParticleModeGravity;
```

重力模式有以下几个独有的属性，这些属性只有在emitterMode被设为kCCParticleModeGravity时才能使用：

```
self.centerOfGravity = CGPointMake(-15, 0);
self.gravity = CGPointMake(-50, -90);
self.radialAccel = -90;
self.radialAccelVar = 20;
self.tangentialAccel = 120;
self.tangentialAccelVar = 10;
self.speed = 15;
self.speedVar = 4;
```

centerOfGravity属性是一个CGPoint，它被用于计算与新生成粒子所处位置之间的位移距离。其实这个名称有点误导人，因为实际的重力中心点应该是新生成粒子的中心点，而centerOfGravity实际上是在离开粒子一段距离的位置上。

gravity属性用于决定所有粒子在X和Y轴的加速度大小。要想让重力中心点有任何效果，粒子的gravity值不应设置的太大，centerOfGravity也不应该被放到离粒子中心点太远的地方。上述代码中的centerOfGravity和gravity值是很好的参考点，你可以在它们基础上进行调节。

如果radialAccel的值是正数，粒子离开发射器越远，加速就越快。如果此属性的值为负数，粒子离开发射器越远，速度就越慢。

tangentialAccel属性同radialAccel类似，只是它可以让粒子围着发射器旋转，粒子旋转离开发射器越远，速度越快。属性值为正时，粒子沿反时针方向旋转；属性值为负时，粒子则沿顺时针方向旋转。

speed属性的作用应该很明显－就是粒子的移动速度。这里的速度没有指定的计量单位。

图9-3 展示了使用重力模式生成的粒子效果。

图9-3. ParticleEffects02项目中使用重力模式生成的粒子效果

发射器模式：半径（Radius）

半径模式会让粒子沿着一个圆形旋转。你也可以生成螺旋状粒子效果，得到漩涡效果或者是螺旋状上升效果。以下代码用于设定半径模式：

```
self.emitterMode = kCCParticleModeRadius;
```

和重力模式一样，半径模式也有一些独有的属性。这些属性只有在emitterMode被设为kCCParticleModeRadius时才能使用：

```
self.startRadius = 100;
self.startRadiusVar = 0;
self.endRadius = 10;
self.endRadiusVar = 0;
self.rotatePerSecond = -180;
self.rotatePerSecondVar = 0;
```

startRadius属性是粒子效果的节点位置和发射粒子的位置之间的距离。endRadius属性则是粒子效果的节点位置和粒子最终要到达的位置之间的距离。如果你想得到完美的粒子圆圈，你可以通过使用以下常量将endRadius和startRadius设置为相同的值：

```
self.endRadius = kCCParticleStartRadiusEqualToEndRadius;
```

你可以使用rotatePerSecond属性来影响粒子移动的方向和速度。如果

startRadius和endRadius的值不相同的话，也会影响粒子沿着圆圈旋转的圈数。

图9-4展示了半径模式下与图9-3在重力模式下得到的相同的粒子效果，它们看上去完全不一样，虽然所有属性的值都设置成相同的 - 除了那些两个模式各自独有的属性。你可以把ParticleEffects02项目中的以下代码去掉注释用的斜杠来进行测试：

```
//self.emitterMode = kCCParticleModeRadius;
```

图9-4. 尽管使用了相同的属性设置（除了模式独有的属性外），与图9-3相同的粒子效果在半径模式下看上去完全不一样

粒子位置

通过移动粒子节点的位置，你也可以移动整个效果。不过效果本身也有一个posVar属性用于控制新粒子将会被生成的位置变化。默认情况下，position和positionVar都在节点的中心位置：

```
self.position = CGPointZero;
self.posVar = CGPointZero;
```

粒子位置一个很重要的方面是：已存在粒子是相对与粒子节点的移动而移动呢？还是它们完全不受节点位置的影响？例如，如果你生成了一圈围着玩家角色的头旋转的星星的粒子特效，你可能需要这圈星星随着角色的移动而移动。你可以通过以下代码实现：

```
self.positionType = kCCPositionTypeGrouped;
```

从另一方面来说，如果你想在玩家角色的身上放火，需要生成一个在玩家移动的时候出现的烟雾拖尾效果，你可以通过以下代码实现：

```
self.positionType = kCCPositionTypeFree;
```

第二种自由移动类效果最适合用在蒸汽，火焰，引擎产生的废气这类效果上。这类效果要求粒子们围绕着物体四周移动，但是又要看起来同发射出这些粒子的物体没有任何的联系。

粒子大小

startSize和endSize属性用于控制粒子的大小，大小是使用像素（pixel）来衡量的。startSize是粒子刚被发射时的大小，endSize是粒子消失时的大小。粒子的大小从startSize渐渐变为endSize。

```
self.startSize = 40.0f;
self.startSizeVar = 0.0f;
self.endSize = kCCParticleStartSizeEqualToEndSize;
self.endSizeVar = 0;
```

kCCParticleStartSizeEqualToEndSize这个常量用于确保粒子的生命期内的大小都不会变化。

粒子方向

发射粒子时粒子射出的方向是用angle属性来控制的。angle为0意味着会向上发射粒子，不过只有在重力模式下才会这样。在半径模式下，angle属性用于决定粒子在startRadius属性上的发射点；angle的值越大，发射点就会沿着圆圈上的反时针方向移动。

```
self.angle = 0;
self.angleVar = 0;
```

粒子生命期

life属性决定着粒子会在几秒钟之内从生成到消失。此属性为单个粒子设置生命期。请记住，粒子的生命期越长，任一时间屏幕上同时存在的粒子就越多。如果屏幕上同时显示的粒子数量达到了最大允许的数量，在一些已存在的粒子消失之前，新粒子将不再被生成。

```
self.life = 5.0f;
self.lifeVar = 1.0f;
```

emissionRate属性直接影响着每秒钟内可以生成的粒子数量。与totalParticles属性相配合，它对粒子效果的外观有很大的影响。

```
self.emissionRate = 30;
self.totalParticles = 250;
```

通常，你需要使用emissionRate来平衡粒子的生命期和totalParticles属性。你可以通过以下公式来计算emissionRate：

```
self.emissionRate = self.totalParticles / self.life;
```

技巧：通过调整粒子的生命期（life），粒子总数（totalParticles）和emissionRate，你可以创造出爆炸效果。原理是：屏幕上的粒子总数是有限的，而新粒子会在短时间内生成，导致粒子的流动被经常中断。从另一方面来说，如果你注意到粒子流动过程中产生了缺口，你可以通过增加粒子数量或者减小生命期（或者emisisonRate）来解决问题。

粒子颜色

每个粒子都可以从一个起始颜色渐变到一个最终颜色，从而创造出艳丽的颜色粒子效果。你至少需要设置startColor，因为默认情况下粒子是黑色的，所以如果不设置起始颜色的话，你有可能看不到粒子。颜色的类型是ccColor4F，这是一个由4个浮点数组成的结构：r，g，b和a，对应红，绿，蓝和透明度。浮点数的数值在0到1之间，1代表着最饱和的颜色。如果你需要完全白色的粒子颜色的话，你应该把所有的r，g，b和a都设为1。

```
startColor.r = 1.0f;
startColor.g = 0.25f;
startColor.b = 0.12f;
startColor.a = 1.0f;
startColorVar.r = 0.0f;
startColorVar.g = 0.0f;
startColorVar.b = 0.0f;
startColorVar.a = 0.0f;
endColor.r = 0.0f;
endColor.g = 0.0f;
endColor.b = 0.0f;
endColor.a = 1.0f;
endColorVar.r = 0.0f;
endColorVar.g = 0.0f;
endColorVar.b = 1.0f;
endColorVar.a = 0.0f;
```

粒子混合模式（Particle Blend Mode）

“混合”是指一个粒子的像素被显示在屏幕上之前所需要经历的计算过程。BlendFunc这个属性以ccBlendFunc结构体（struct）作为输入，ccBlendFunc提供了来源混合模式和目标混合模式：

```
self.blendFunc = (ccBlendFunc){GL_SRC_ALPHA, GL_DST_ALPHA};
```

“混合”的工作方式是在粒子渲染的时候，将源图片（粒子）的红，绿，蓝和透明度信息与屏幕上已经存在的图片颜色信息相混合。实际效果是，粒子和它所处的背景以某种方式进行了混合，而blendFunc则决定着源图片颜色和背景颜色的混合程度。

BlendFunc属性对粒子的外观有很大的影响。通过在来源和目标设置中使用下面的混合模式，你可以生成非常奇特的效果。你可以用它们做很多不同的尝试：

- GL_ZERO
- GL_ONE
- GL_SRC_COLOR
- GL_ONE_MINUS_SRC_COLOR
- GL_SRC_ALPHA
- GL_ONE_MINUS_SRC_ALPHA

- GL_DST_ALPHA
- GL_ONE_MINUS_DST_ALPHA

你可以通过OpenGL ES文档，找到更多关于OpenGL混合模式的信息和混合模式计算的细节信息，以下是网址：
www.khronos.org/opengles/documentation/opengles1_0/html/glBlendFunc.html

来源混合模式GL_SRC_ALPHA和目标混合模式GL_ONE经常配合使用来生成递增型的混合效果。当很多粒子相互叠加时，得到的结果是非常亮的颜色，甚至是白色：

```
self.blendFunc = (ccBlendFunc){GL_SRC_ALPHA , GL_ONE};
```

另外，你可以将blendAdditive属性设为YES，这和把blendFunc设置为GL_SRC_ALPHA和GL_ONE的效果是一样的：

```
self.blendAdditive = YES;
```

如果将混合模式设置为GL_SRC_ALPHA和GL_ONE_MINUS_SRC_ALPHA的话，得到的将会是透明的粒子：

```
self.blendFunc = (ccBlendFunc){GL_SRC_ALPHA , GL_ONE_MINUS_SRC_ALPHA};
```

粒子贴图

如图9-2所示，如果没有贴图的话，所有粒子将会是单调的色块。要在粒子效果中使用贴图，你可以使用CCTextureCache的addImage方法，此方法会返回一个由指定贴图生成的CCTexture2D节点：

```
self.texture = [[CCTextureCache sharedTextureCache] addImage:@"fire.png"];
```

用于粒子效果的贴图最好是云状的，粗略看上去像球形的图片。如果贴图有高对比度区域的话，通常对生成的粒子效果是有害的。比如说在之前射击游戏中使用过的redcross.png图片（红十字图片），这样的图片可以让人看到单独的粒子，因为这样的粒子很难互相混合。不过有些情况下倒是有用的，比如之前提到过的围着角色头部旋转的一圈星星。

粒子效果贴图最重要的一点是图片尺寸不能超过64x64像素。贴图尺寸越小，粒子效果就运行的越流畅。

介绍Particle Designer（粒子设计器）

Particle Designer是一个用于为cocos2d和iOS的OpenGL程序设计和生成粒子效果的工具。你可以在以下网址下载试用版：
http://particledesigner.71squared.com.

使用这个工具，你可以在创造粒子效果时省去很多时间。你可以在改变粒子属

性的同时实时地看到粒子效果的变化。默认情况下，软件的用户界面展示了一个粒子效果的列表。如果需要修改某个粒子效果，你可以选中它，然后通过双击选中的粒子效果或者点击右上角的Emitter Config按钮，切换到Emitter Config视图（图9-5）。

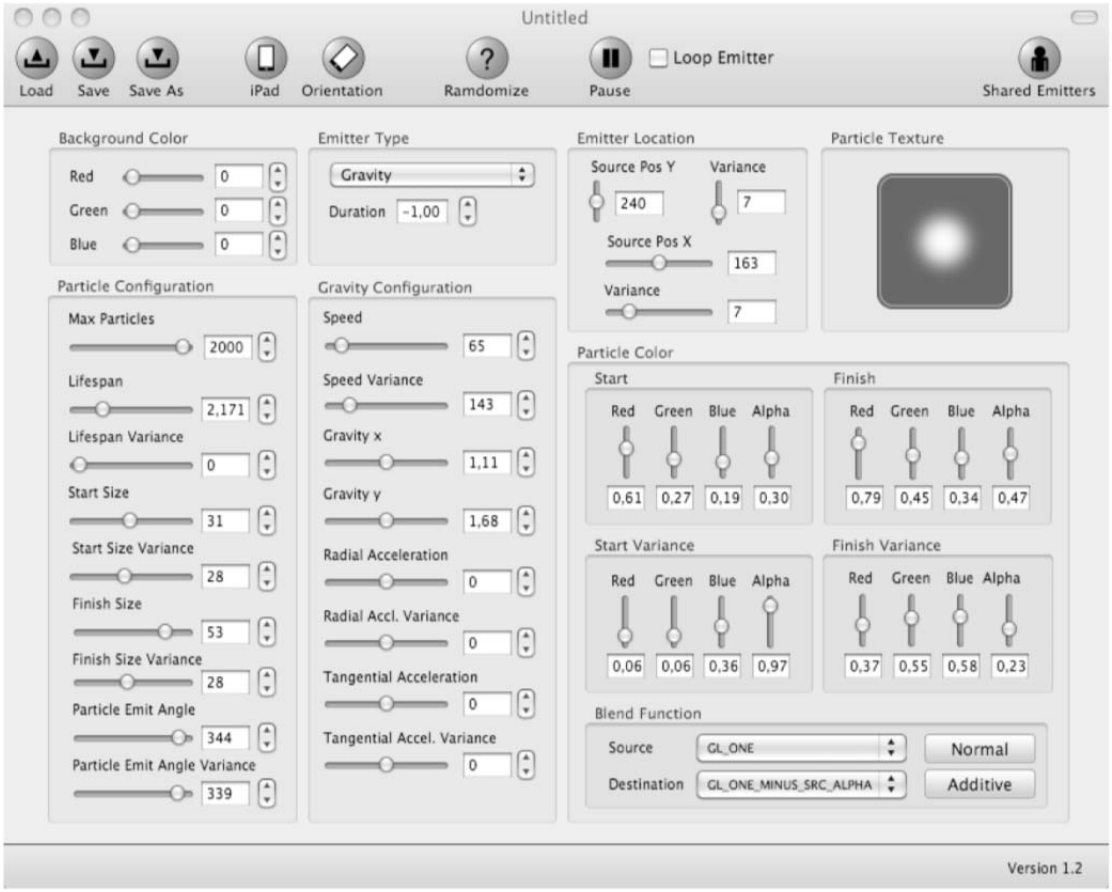

图9-5. 你可以在Particle Designer中修改各种参数。有另一个窗口（本图中没有显示）会实时显示修改后的粒子效果。

你应该认得出界面上的这些参数。只有少数几个我们之前讨论过的参数不能通过Particle Designer进行编辑。一个叫做positionType，另一个是在半径发射器模式下的endRadiusVar。不能编辑endRadiusVar意味着你不能通过软件创建在半径模式下向外旋转的粒子效果。不过你可以在代码中加载完成的粒子效果，然后直接在代码里修改相关的属性。与可以在软件界面上实时看到生成的粒子效果相比，不能修改一两个参数所带来的不便是微不足道的。

软件界面上唯一一个不同寻常的地方是粒子贴图的加载。软件界面上没有用于加载图片的按钮，双击图片的区域也没有任何效果。它的秘密是：你只能把需要的图片拖动到粒子贴图的区域，软件只接受这样的方式来加载粒子贴图。从Finder中拖动一张图片到粒子贴图区域，那块区域会变成绿色，表示软件将会接受图片。一旦软件接受了图片，当前粒子效果将会使用新加载的图片。**注意**：如果你使用超过64x64像素的图片，Particle Designer会警告你，不过它还是会使用你提供的图片，只是会自动将其缩小到64x64，而且不会考虑图片的原始长宽比。

图9-6展示了Particle Designer的预览窗口，看起来和iPhone模拟器几乎一样。你也可以将其设置为iPad的屏幕尺寸，而且可以通过点击软件菜单栏中的iPad或iPhone按钮和Orientation按钮（在Load，Save和Save As按钮的右边）来改变窗口的显示方向。你可以在预览窗口中拖动粒子效果，这样就可以看到粒子效果移动时的效果了。

Background Color（背景色）的设置是和粒子效果的设置分开的。此设置会改变预览窗口中的背景色。

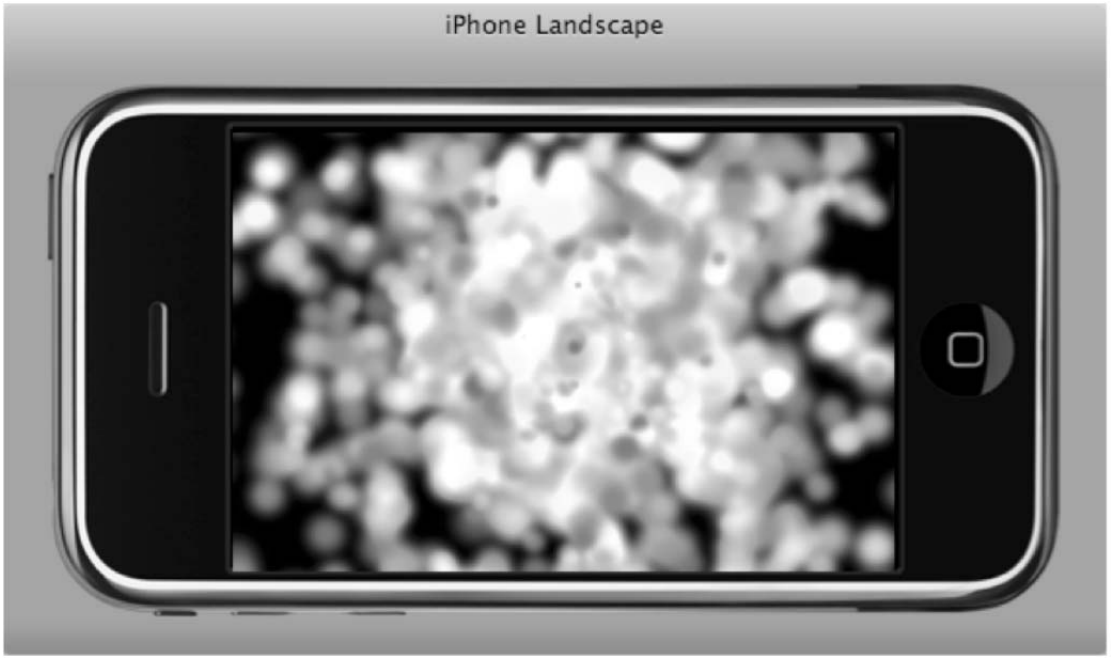

图9-6. Particle Designer拥有一个和iPhone模拟器一样的预览窗口。你也可以将其显示尺寸设置为iPad的屏幕大小。如果你点击和拖动屏幕里的粒子效果，你也可以看到它们在移动时的样子。

如果你缺乏灵感，你可以使用软件界面顶部中央的“Ramdomize”按钮。虽然这个按钮不会把所有的参数值进行随机化，比如，“Ramdomize”不会改变Emitter Type（发射器类型）， Emitter Location（发射器位置）和很多与Emitter Type相关的参数，但是生成的随机粒子效果还是可以给你很多灵感的。

一旦你找到灵感，你就可以在软件界面上通过改变各个参数，查看预览窗口中的实时效果了。慢慢的你就会找到你想要的效果了。不过你要小心，这个调整参数的过程很让人着迷，你很可能不知不觉的在上面花去好多时间。

警示：设计粒子效果的时候要注意以下几点。首先，你要时刻记住，你的游戏除了渲染粒子效果之外还要渲染很多其它的东西。如果你在Particle Designer中设计的粒子效果在它的预览窗口中是以60帧每秒的速度运行的，并不意味着在你游戏中实际运行时也可以达到60帧每秒。所以你一定要在实际的游戏中测试粒子效果。

其次，你要在真实的iOS设备上进行测试。在iOS模拟器中测试游戏所得到的结果很多时候是误导人的，甚至和实际设备上运行的结果完全不一样。

使用Particle Designer生成的粒子效果

在Paricle Designer中完成粒子效果设计以后，接下去就是要把它用到cocos2d游戏中去。首先你需要保存粒子效果，在Particle Designer界面上点击 Save 或者 Save As 按钮，你会得到如**图9-7**所示的对话框：

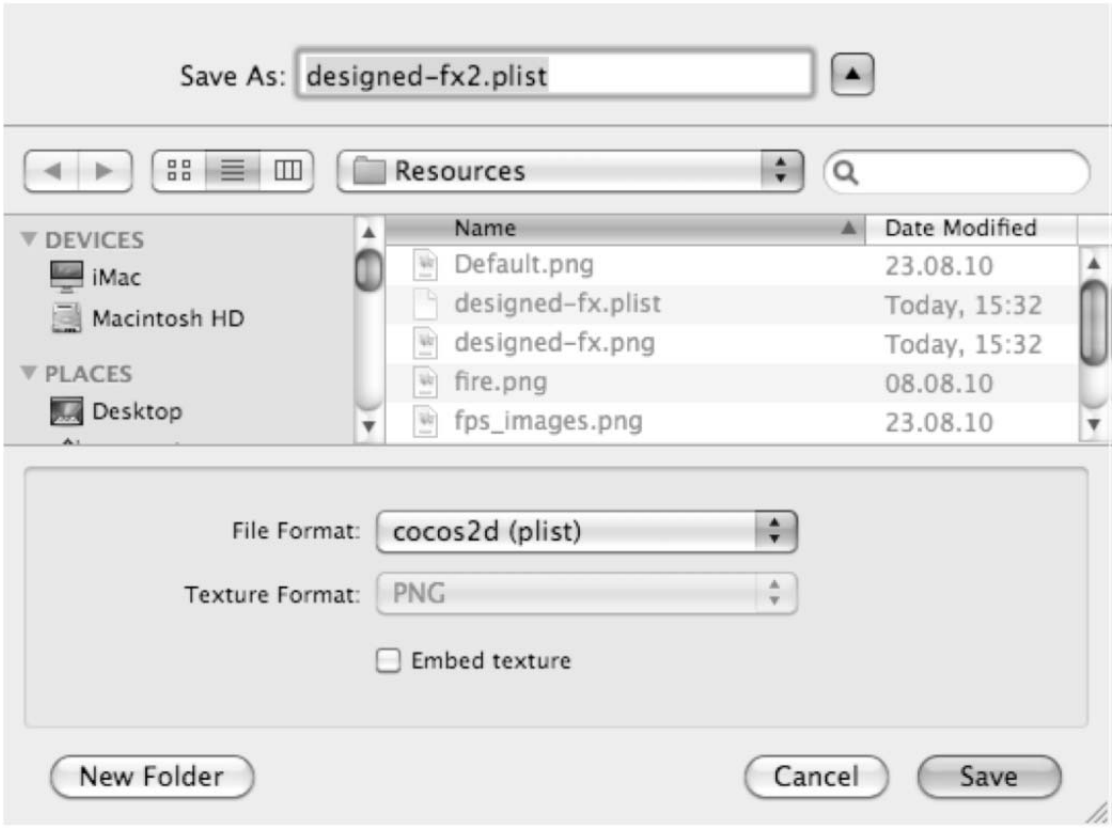

图9-7. 保存Particle Designer中生成的粒子效果时，你需要将File Format设为cocos2d(plist)。Embed texture（将贴图嵌入plist文件）这个选项是可选的。

为了在cocos2d中使用生成的粒子效果，你必须将输出的File Format（文件格式）设置为cocos2d(plist)。你可以勾选Embed Texture选项，这样会把粒子贴图保存到plist文件中。这样做的好处是你只需要将plist文件添加到Xcode项目中就可以了；坏处是如果你要使用不同的粒子贴图，你必须使用Particle Designer把旧的贴图用新的贴图替换掉，重新导出plist文件，然后重新添加到Xcode项目中。

完成粒子效果的保存以后，你可以将导出的plist文件添加到Xcode项目中。如果你没有将贴图嵌入到plist文件中，你还需要将贴图的PNG文件添加到Xcode项目的Resources组中。我创建了ParticleEffects03项目，在这个项目中我添加了两种plist文件，一种嵌入了贴图文件，另一种没有嵌入。

列表9-4展示了修改过的runEffect方法，允许使用Particle Designer生成的粒子效果了。

列表9-4. 使用Particle Designer生成的粒子效果

```
-(void) runEffect
```

```
{
        // 移除之前使用的粒子效果
        [self removeChildByTag:1 cleanup:YES];

        CCParticleSystem* system;

        switch (particleType)
        {
                case ParticleTypeDesignedFX:
                        system = [CCQuadParticleSystem particleWithFile:@"fx1.plist"];
                        break;
                case ParticleTypeDesignedFX2:
                        system = [CCQuadParticleSystem particleWithFile:@"fx2.plist"];
                        system.positionType = kCCPositionTypeFree;
                        break;
                case ParticleTypeSelfMade:
                        system = [ParticleEffectSelfMade node];
                        break;
                default:
                        // 什么都不做
                        break;
        }

        CGSize winSize = [[CCDirector sharedDirector] winSize];
        system.position = CGPointMake(winSize.width / 2, winSize.height / 2);
        [self addChild:system z:1 tag:1];

        [label setString:NSStringFromClass([system class])];
}
```

上述代码中，通过使用 particleWithFile 方法和Particle Designer中生成的粒子效果plist文件，我们将CCParticleSystem进行初始化。我在这里选择了CCQuadParticleSystem，因为它适用于所有的iOS设备。你也可以通过使用ARCH_OPTIMAL_PARTICLE_SYSTEM关键词让cocos2d来决定应该使用哪个粒子系统。代码如下：

```
system = [ARCH_OPTIMAL_PARTICLE_SYSTEM particleWithFile:@"fx1.plist"];
```

警示：在使用Particle Designer生成的粒子效果时，你必须使用CCQuadParticleSystem或者CCPointParticleSystem进行粒子效果的初始化。虽然CCParticleSystem作为父类也实现了particleWithFile这个方法，但是如果你使用父类初始化粒子效果的话，屏幕上将不会显示任何东西。

我把用于将粒子系统放置在屏幕中央的那两行代码移出了switch判断，以避免代码重复。如我之前提到过的，我们要尽量避免复制粘贴重复的代码。

分享粒子效果

Particle Designer的一个很酷的地方是你可以与别的用户分享创造出来的粒子效果。在软件的菜单栏，选择 Share，然后点击 Share Emitter，如图9-8所示，会出现一个对话框，你可以在里面输入将要分享的粒子效果标题和说明。

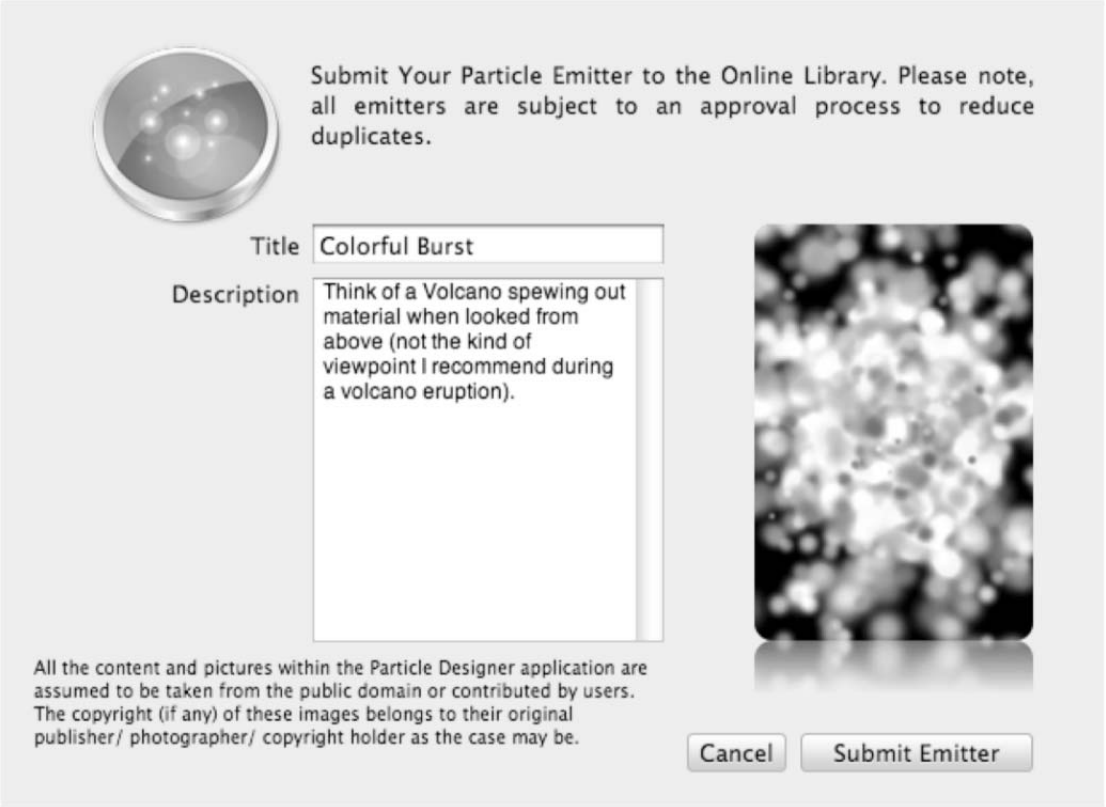

图9-8. 你可以提交自己创造的粒子效果到Particle Designer的在线仓库，与别的用户分享自己的创造物。

图9-9中，你可以看到我提交的粒子效果处于右下角处。

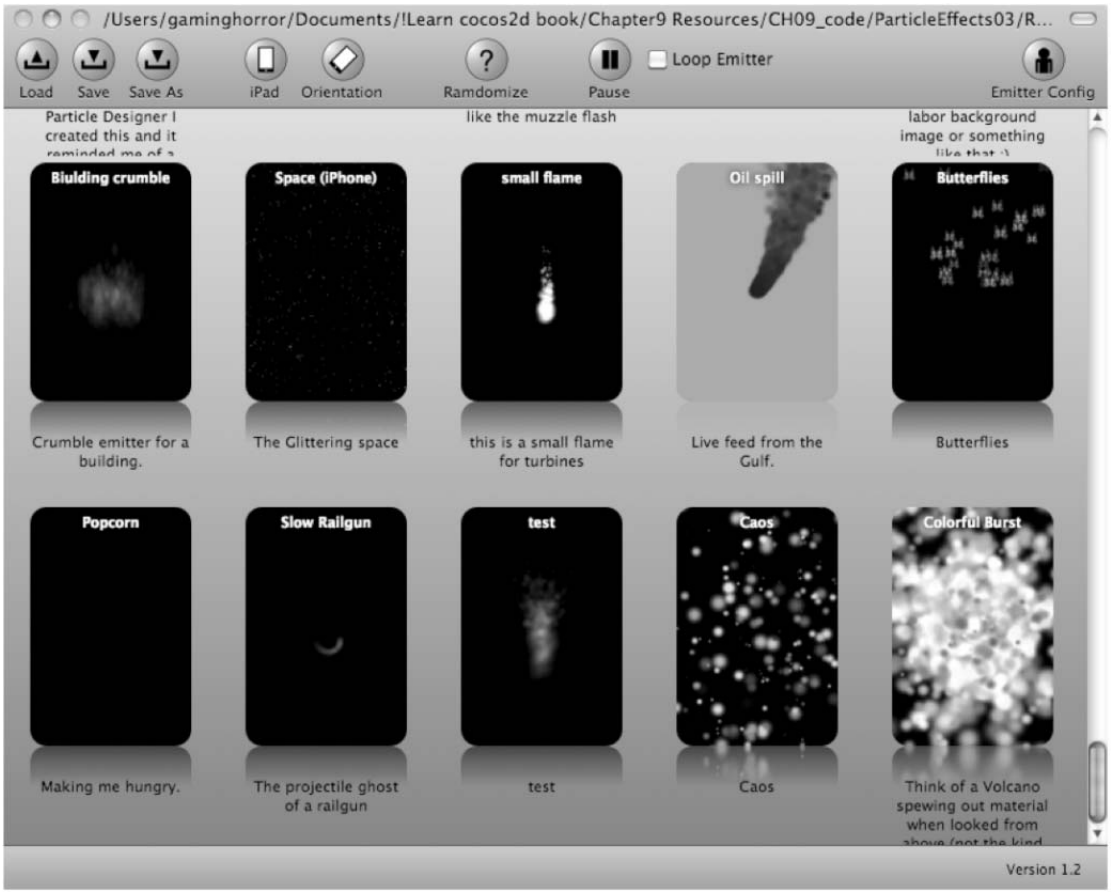

图9-9. 提交的粒子效果很快就会出现在在线仓库中。我刚刚提交的粒子效果就在右下角处。很显然，我的描述太长了，不能完全显示在屏幕上。

别人分享的粒子效果并不总能够满足你的需求，不过你通常可以找到类似的效果作为起始点。它们可以帮助你更快的制作出满足你需求的效果，至少可以为你提供灵感。

在射击游戏中添加粒子效果

让我们在之前制作的射击游戏中添加一些粒子效果。如图9-10所示，完成的最终效果可以在本章的ShootEmUp04项目中找到。

如列表9-5代码所示，EnemyEntity类中的gotHit方法是添加毁灭式粒子效果最好的地方。我决定给Boss怪物添加它独有的粒子效果-因为BOSS很大，而且是紫色的。

列表9-5. 为Shoot’em Up游戏添加爆炸效果

```
-(void) gotHit
{
        hitPoints--;
        if (hitPoints <= 0)
        {
                self.visible = NO;
```

```
            // 当敌人被消灭时，播放粒子效果
            CCParticleSystem* system;

            if (type == EnemyTypeBoss)
            {
                system = [ARCH_OPTIMAL_PARTICLE_SYSTEM
                particleWithFile:@"fx-explosion2.plist"];
            }
            else
            {
                system = [ARCH_OPTIMAL_PARTICLE_SYSTEM
                particleWithFile:@"fx-explosion.plist"];
            }

            // 在此设置无法在Particle Designer中修改的一些参数
            system.positionType = kCCPositionTypeFree;
            system.autoRemoveOnFinish = YES;
            system.position = self.position;

            // 将粒子效果添加到GameScene
            [[GameScene sharedGameScene] addChild:system];
        }
}
```

fx-explosion.plist和fx-explosion2.plist这两个粒子效果文件必须作为资源添加到Xcode项目中。粒子系统的初始化和之前一样。因为粒子效果必须独立于生成它的敌人对象，我们需要做些参数调整。首先，autoRemoveOnFinish要设置为YES，这样粒子效果在完成播放后就会自动将自己移除。其次，粒子效果需要知道敌人的当前位置，以便在敌人的位置显示粒子效果。

因为敌人本身不能显示粒子效果，所以我把生成的粒子效果添加到了GameScene里面。敌人在一开始的时候是隐形的，而且它可以随时被重新激活（变成可视的），对粒子效果产生干扰。但是最重要的一个原因是：EnemyEntity对象是被添加到CCSpriteBatchNode中的，而你只能向CCSpriteBatchNode添加CCSprite对象。所以，如果将粒子效果添加到EnemyEntity对象中的话，程序会抛出异常。

现在，当你玩游戏的时候，你可能会注意到在粒子效果显示之前，游戏会停滞一小段时间。那是因为cocos2d在加载粒子效果需要的贴图文件 - 这是个很缓慢的过程。这与贴图是否被嵌入plist文件无关。为了避免发生上述情况，我们可以在GameScene中添加一个预加载的机制：在init方法中为每一个需要在游戏过程中用到的粒子效果调用preloadParticleEffect（预加载粒子效果）方法：

```
// 通过在屏幕外播放每个要用到的粒子效果，完成粒子贴图的加载操作
[self preloadParticleEffects:@"fx-explosion.plist"];
[self preloadParticleEffects:@"fx-explosion2.plist"];
```

preLoadParticleEffect方法所做的就是生成粒子效果。因为返回的对象是一个自动释放对象，所以它们的内存会被自动释放。但是加载过的贴图会被保留在CCTextureCache中：

```
-(void) preloadParticleEffects:(NSString*)particleFile
{
	[ARCH_OPTIMAL_PARTICLE_SYSTEM particleWithFile:particleFile];
}
```

如果你没有将贴图嵌入粒子效果的plist文件中，你可以通过调用CCTextureCache的addImage方法来预加载粒子效果的贴图：

```
[[CCTextureCache sharedTextureCache] addImage:particleFile];
```

图9-10. Boss被杀以后的粒子效果

结语

本章讨论的都是视觉上的东西。cocos2d自带的粒子效果为我们展示了各种可能性。

不过直接通过代码来生成粒子效果是件很痛苦的事情。因为有很多属性需要调整；有些还是不同的发射器模式所独有的；还有些属性的命名容易迷惑人，导致误解。不过通过解释每一个属性，现在你应该已经了解粒子效果是如何生成的，哪些属性是最重要的。

我们学习了如何使用Particle Designer生成粒子效果。这是一个很有用的工具，也很有趣。你可以通过软件界面直接拖动滚动条来改变各种属性的值，实时的在预览屏幕上看到粒子效果的更新。你可以通过Particle Designer设计需要的粒子效果，更可以与别的软件用户分享你创造的效果，并且可以让别人分享出来的效果为己所用。

最后的部份，我们讨论了如何给上一章中的射击游戏加上粒子效果。现在当敌人被消灭时，就会播放已经加载的粒子效果。这让我们的游戏看上去更加的有趣。

在下一章中，我将把射击游戏放到一边，专门介绍一下“瓷砖地图”（tilemap）的知识。