# 第5章 Cocos2D中的文本渲染系统

第4章学习了Cocos2D强大的动作、特效和动画效果。本章将要学习Cocos2D中相对静态的一面，即文本渲染系统。Cocos2D支持所有内置的iOS字体，以及一些TrueType字体。使用Cocos2D强大的文本渲染系统，可以在应用和游戏中轻松的添加标签和文本。本章示例项目以前4章的项目为基础，此外提供额外的示例项目，以帮助读者更好地掌握相关内容。

## 文本渲染系统的组成

在Cocos2D中，文本渲染系统由两个重要的类实现：CCLabelTTF和CCLabelBMFont。此外还有一个不太常用的CCLabelAtlas类。本节将详细说明这三个类，以及如何在实际的项目中使用。

### CCLabelTTF类

在使用Cocos2D开发的项目中，经常用CCLabelTTF类显示一些静态的标签和文本。CCLabelTTF类继承自CCSprite，主要用于渲染文本标签。使用CCLabelTTF类可以用很少的代码将文本嵌入到游戏之中。之前显示游戏分数和玩家生命值时，也使用到了CCLabelTTF类。

本节将详细介绍该类的定义及其在项目中的使用。

#### 认识CCLabelTTF类

在安装Cocos2D的目录中找到cocos2d-ios.xcodeproj，双击打开该项目。在Cocos2D文件夹下的Label Nodes中找到CCLabelTTF.h，单击该文件名可以看到CCLabelTTF类的定义，如图5-1所示。

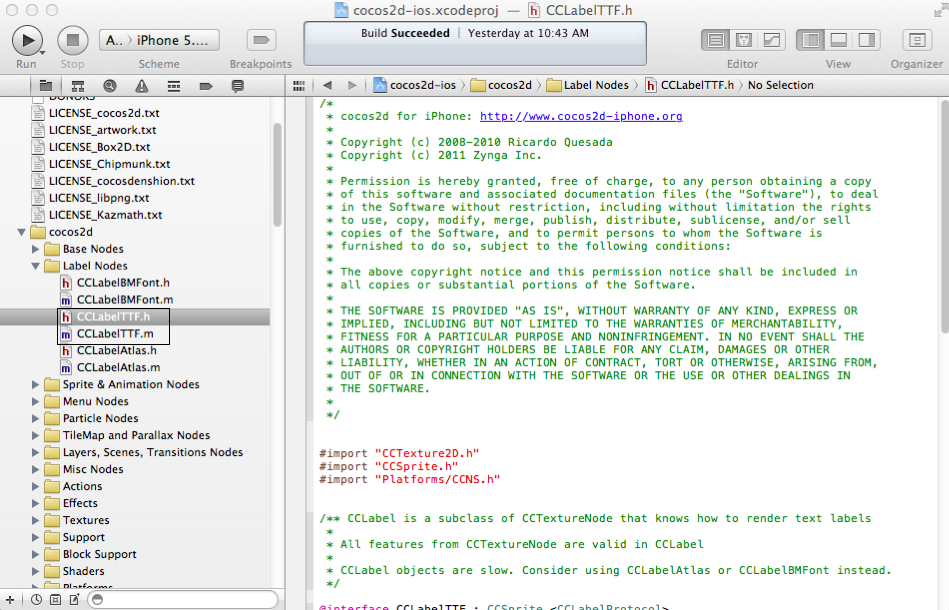


图5-1在Xcode中查看CCLabelTTF.h的定义

首先详细看该类的定义如代码清单5-1所示。

1. CCLabelTTF类的定义

#import "CCTexture2D.h"

#import "CCSprite.h"

#import "Platforms/CCNS.h"

/\*\* CCLabel is a subclass of CCTextureNode that knows how to render text labels

\*

\* All features from CCTextureNode are valid in CCLabel

\*

\* CCLabel objects are slow. Consider using CCLabelAtlas or CCLabelBMFont instead.

\*/

@interface CCLabelTTF : CCSprite <CCLabelProtocol>

{

CGSize dimensions\_;

CCTextAlignment alignment\_;

NSString \* fontName\_;

CGFloat fontSize\_;

CCLineBreakMode lineBreakMode\_;

NSString \*string\_;

}

/\*\* creates a CCLabel from a fontname, alignment, dimension in points, line break mode, and font size in points.

Supported lineBreakModes:

- iOS: all UILineBreakMode supported modes

- Mac: Only NSLineBreakByWordWrapping is supported.

@since v1.0

\*/

+ (id) labelWithString:(NSString\*)string dimensions:(CGSize)dimensions alignment:(CCTextAlignment)alignment lineBreakMode:(CCLineBreakMode)lineBreakMode fontName:(NSString\*)name fontSize:(CGFloat)size;

/\*\* creates a CCLabel from a fontname, alignment, dimension in points and font size in points\*/

+ (id) labelWithString:(NSString\*)string dimensions:(CGSize)dimensions alignment:(CCTextAlignment)alignment fontName:(NSString\*)name fontSize:(CGFloat)size;

/\*\* creates a CCLabel from a fontname and font size in points\*/

+ (id) labelWithString:(NSString\*)string fontName:(NSString\*)name fontSize:(CGFloat)size;

/\*\* initializes the CCLabel with a font name, alignment, dimension in points, line brea mode and font size in points.

Supported lineBreakModes:

- iOS: all UILineBreakMode supported modes

- Mac: Only NSLineBreakByWordWrapping is supported.

@since v1.0

\*/

- (id) initWithString:(NSString\*)str dimensions:(CGSize)dimensions alignment:(CCTextAlignment)alignment lineBreakMode:(CCLineBreakMode)lineBreakMode fontName:(NSString\*)name fontSize:(CGFloat)size;

/\*\* initializes the CCLabel with a font name, alignment, dimension in points and font size in points \*/

- (id) initWithString:(NSString\*)string dimensions:(CGSize)dimensions alignment:(CCTextAlignment)alignment fontName:(NSString\*)name fontSize:(CGFloat)size;

/\*\* initializes the CCLabel with a font name and font size in points \*/

- (id) initWithString:(NSString\*)string fontName:(NSString\*)name fontSize:(CGFloat)size;

/\*\* changes the string to render

\* @warning Changing the string is as expensive as creating a new CCLabel. To obtain better performance use CCLabelAtlas

\*/

- (void) setString:(NSString\*)str;

@end

熟悉Objective-C的开发者可以从上面代码了解到，CCLabelTTF类直接继承自CCSprite，同时实现了CCLabelProtocol协议。这个协议内容很少，只需知道它提供Label（文本标签）的通用接口，其定义可参考CCProtocol.h的第101行。

CCLabelTTF提供3种用于初始化标签的类方法和实例方法，同时还提供了一种方法用于设置文本内容。其中支持换行模式设置的方法是v1.0版本后添加的新方法。

如果读者想了解每种方法是如何具体实现的，可以在Xcode中切换到CCLabelTTF.m，查看更深入的内容。

#### 如何使用CCLabelTTF类

有了以上的理论基础，接下来看看如何在项目中实际使用CCLabelTTF。

1. 在Xcode中创建一个新项目。

打开Xcode，单击File→New，选择New Project，在iOS下选择Cocos2D，选择Cocos2D标准模板，单击Next，在Product Name处填上LabelTTF，选择存储的位置，单击Create。

在Xcode中切换到HelloWorldLayer.m，在init方法中会看到默认代码中的第一行，其作用是使用CCLabelTTF创建一个标签对象，如代码清单5-2所示。

1. 使用CCLabelTTF创建一个标签对象

// create and initialize a Label

CCLabelTTF \*label = [CCLabelTTF labelWithString:@"Hello World" fontName:@"Marker Felt" fontSize:64];

在以上代码中，通过类方法labelWithString:fontName:fontSize:，使用特定的文本内容、字体名称和字体大小这三个参数创建并初始化一个新的标签对象。通过该方法，CCLabelTTF对象将使用CCTexture2D类从文本创建一个纹理图，并在屏幕上显示该纹理图（具体实现方法请参考CCLabelTTF.m）。

提示 事实上，所有使用Cocos2D标准模板创建的新项目默认情况下都带有这行代码。

一旦创建完CCLabelTTF对象，就可以像使用其他CCNode节点一样使用它们，比如将其添加到当前层，设置它的位置，等等。对于静态文本，使用CCLabelTTF类是再方便不过了。

注意 每次调用setText方法时，都会创建一个新的纹理图。因此，如果在程序中需要每帧都修改文本内容，使用该方法将会对游戏性能产生很大的影响。可以使用后面介绍的CCLabelBMFont类来提高游戏性能。

1. 修改init方法。

修改HelloWorldLayer.m的init方法如代码清单5-3所示。

1. 修改后的init方法

-(id) init

{

// always call "super" init

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

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

// 创建并初始化标签对象

CCLabelTTF \*label1 = [CCLabelTTF labelWithString:@"We Love Apple!" fontName:@"Marker Felt" fontSize:30];

CCLabelTTF \*label2 = [CCLabelTTF labelWithString:@"We Love iPhone!" dimensions:CGSizeMake(150, 150) alignment:UITextAlignmentRight fontName:@"ArialRoundedMTBold" fontSize:32];

CCLabelTTF \*label3 = [CCLabelTTF labelWithString:@"We Love iPad!" dimensions:CGSizeMake(150, 150) alignment:UITextAlignmentLeft lineBreakMode:UILineBreakModeWordWrap fontName:@"Chalkduster" fontSize:32];

// 获取屏幕大小

CGSize size = [[CCDirector sharedDirector] winSize];

// 设置标签的位置

label1.position = ccp( size.width /2 , size.height/2 );

label2.position = ccp(size.width/2,150+size.height/2);

label3.position = ccp(size.width/2, -150+size.height/2);

// 将标签对象添加为当前层的子节点

[self addChild: label1];

[self addChild:label2];

[self addChild:label3];

//让label对象执行动作

id label1Action = [CCSpawn actions:[CCScaleBy actionWithDuration:5.0 scale:1.3],[CCFadeIn actionWithDuration:5.0], nil];

[label1 runAction:label1Action];

id label2Action = [CCSpawn actions:[CCFadeIn actionWithDuration:5.0],[CCRotateBy actionWithDuration:5.0 angle:180], nil];

[label2 runAction:label2Action];

id label3Action = [CCSpawn actions:[CCFadeIn actionWithDuration:5.0],[CCRotateBy actionWithDuration:5.0 angle:180],nil];

[label3 runAction:label3Action]; }

return self;

}

以上代码进行了以下操作：

1. 创建并使用三种不同的类方法来初始化两个标签对象；
2. 使用CCDirector单例获取屏幕大小；
3. 设置三个标签对象的位置；
4. 将三个标签对象添加为当前层的子节点；
5. 让第一个标签对象执行同步动作，在5秒内缩放到原大小的2倍同时淡入；让第二个标签执行同步动作，在5秒内淡入并旋转180度；让第三个标签执行动作，行同步动作，在5秒内淡入并旋转180度。
6. 编译并运行该项目。

屏幕上出现两个标签，其中“We Love Apple!”以淡入缩放的形式出现，而“We love iPhone!”则以淡入旋转的方式出现在上方，“We Love iPad!”会以淡入旋转的方式出现在下方。它们以三种不同的字体显示，如图5-2所示。



图5-2三种不同字体的标签

该部分代码请参考示例项目LabelTTF。

#### CCLabelTTF类的可用字体

CCLabelTTF类可以用的字体必须是iOS中用到的字体。而苹果在每次推出新的iOS版本时都会增加可用字体。想知道CCLabelTTF中可以用哪些字体，可以查询UIFont类中iOS的可用字体。在init方法中继续添加以下代码可以在控制台中显示所有可用字体，如代码清单5-4所示。

1. 显示iOS中的可用字体

//显示iOS中的可用字体

NSMutableArray \*fontNames = [[NSMutableArray alloc]init];

NSArray \*fontFamilyNames =[UIFont familyNames];

for (NSString \*familyName in fontFamilyNames){

NSLog(@"Font Family Name = %@",familyName);

NSArray \*names = [UIFont fontNamesForFamilyName:familyName];

NSLog(@"Font Names = %@",fontNames);

[fontNames addObjectsFromArray:names];

}

[fontNames release];

编译运行项目，即可在控制台中看到UIFont类返回的字体家族及每个字体家族中所包含的字体，如图5-4所示。

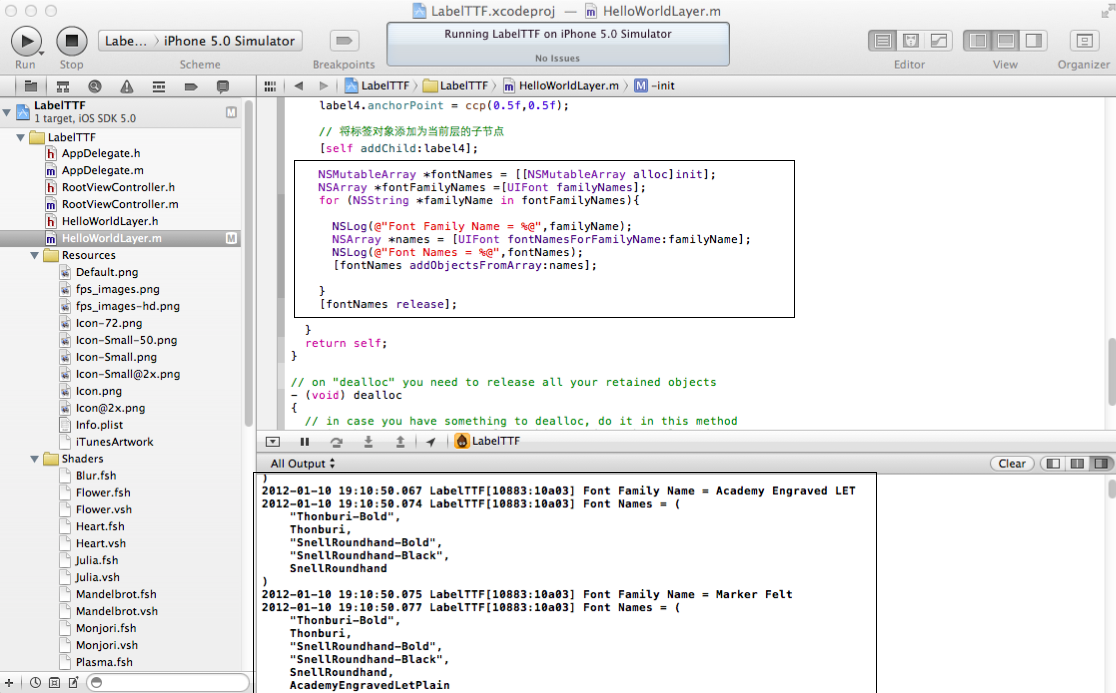


图5-4在控制台中显示iOS所支持的字体列表

此外，可以在http://iosfonts.com/直接查看iOS当前版本所支持的字体种类。

### CCLabelAtlas类

虽然使用CCLabelTTF显示静态标签文本比较方便，但其渲染速度相对较慢，且通常用于显示纯文本。为了提高文本渲染速度，同时为在项目中显示文本提供更大的灵活性，Cocos2D提供了CCLabelTTF的替代类CCLabelAtlas。

#### 认识CCLabelAtlas类

CCLabelAtlas是CCAtlas的子类，它和CCLabelTTF的主要区别在于：

* CCLabelAtlas的渲染速度要远快于CCLabelTTF；
* CCLabelAtlas的字符宽度和高度都是固定的；
* CCLabelAtlas中的字符可以是任何形式，因为它们来自于图片文件。

在安装Cocos2D目录中找到cocos2d-ios.xcodeproj，双击打开该项目。在Cocos2D文件夹下的Label Nodes中找到CCLabelAtlas.h,单击该文件名可以看到CCLabelAtlas类的定义，如图5-5所示。

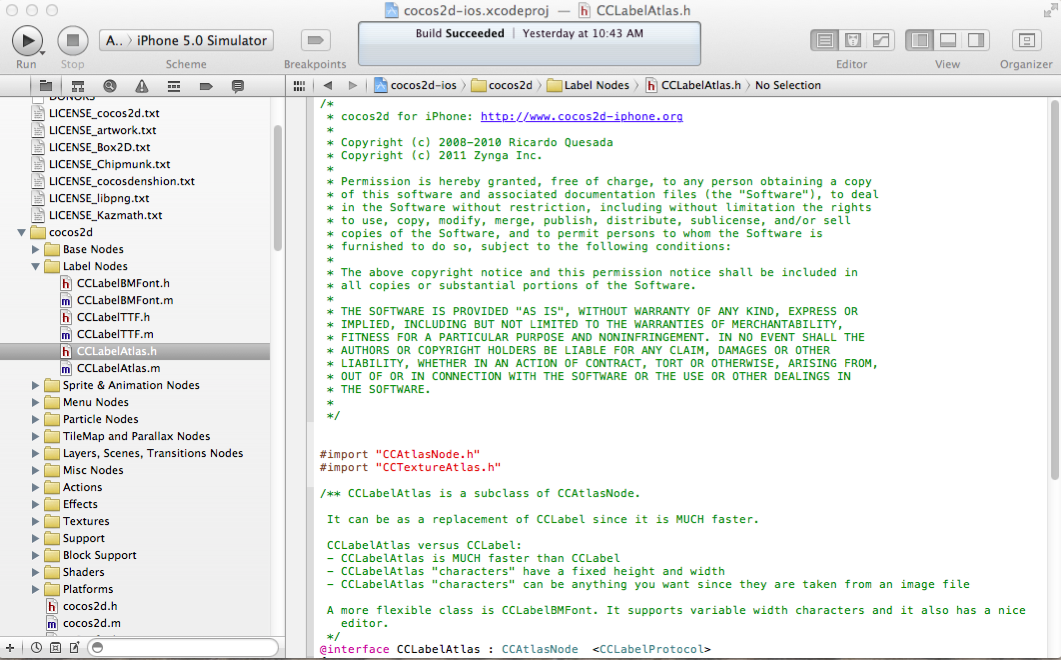


图5-5在Xcode中查看CCLabelAtlas.h的定义

CCLabelAtlas类的定义如代码清单5-5所示。

1. CCLabelAtlas类的定义

@interface CCLabelAtlas : CCAtlasNode <CCLabelProtocol>

{

// string to render

NSString \*string\_;

// the first char in the charmap

unsigned char mapStartChar\_;

}

/\*\* creates the CCLabelAtlas with a string, a char map file(the atlas), the width and height of each element in points and the starting char of the atlas \*/

+(id) labelWithString:(NSString\*) string charMapFile: (NSString\*) charmapfile itemWidth:(NSUInteger)w itemHeight:(NSUInteger)h startCharMap:(unsigned char)c;

/\*\* initializes the CCLabelAtlas with a string, a char map file(the atlas), the width and height in points of each element and the starting char of the atlas \*/

-(id) initWithString:(NSString\*) string charMapFile: (NSString\*) charmapfile itemWidth:(NSUInteger)w itemHeight:(NSUInteger)h startCharMap:(unsigned char)c;

@end

从以上代码中可以看到，CCLabelAtlas类继承自CCAtlasNode，并实现了CCLabelProtocol协议。而CCAtlasNode继承自CCNode，并实现了CCRGBAProtocol和CCTextureProtocol协议。CCLabelAtlas类相对很简单，只有一个用于初始化的类方法和实例方法。

如果读者想了解这些方法是如何具体实现的，可以切换到CCLabelAtlas.m，查看更深入的内容。

#### 如何使用CCLabelAtlas类

有了前面的理论知识，接下来通过示例看看如何在项目中实际使用CCLabelAtlas。

1. 在Xcode中创建一个新项目。

打开Xcode，单击File→New，选择New Project。在iOS下选择Cocos2D，选择Cocos2D标准模板，单击Next。在Product Name处填上LabelAtlas，选择存储的位置，单击Create。

1. 加载资源文件。

把资源文件（chapter5/resources/fonts/tuffy\_bold\_italic-charmap.png）拖到Xcode的Resources中，并确保选中“Copy items into destination group’s folder(if needed)”。

1. 修改init方法。

切换到HelloWorldLayer.m文件，找到init方法，更改其中创建标签的代码，修改后的代码如代码清单5-6所示。

1. init方法的实现代码

-(id) init

{

// always call "super" init

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

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

// 使用CCLabelAtlas来创建label对象

CCLabelAtlas \*label = [CCLabelAtlas labelWithString:@"123Test" charMapFile:@"tuffy\_bold\_italic-charmap.png" itemWidth:48 itemHeight:64 startCharMap:' '];

// 获取屏幕的大小

CGSize size = [[CCDirector sharedDirector] winSize];

// 将标签放置在屏幕的中心位置

label.position = ccp( 0, size.height/2 );

// 将标签添加为当前层的子节点

[self addChild: label];

}

return self;

}

以上代码，唯一陌生的就是创建label对象的那行代码。使用该方法，需要指定宽度和高度作为charMapFile的参数。同时，其初始位置需要一个字符参数，即startCharMap，但通常情况下其实际的开始值是字符的ASCII码值。在这里指定的是代表空格的ASCII码值32。根据所提供的字符集图片，系统会按照提供的ASCII值去图片上截取所需要的字符出来。因为这里指定的图片开始位置是空格，所以会按照先后顺序在图片中截取。编译运行项目，将看到如图5-6所示的效果。



图5-6使用CCLabelAtlas所创建的标签

注意 CCLabelAtlas的使用相对复杂。在实际的开发中，通常会使用CCLabelBMFont来代替CCLabelAtlas。

### CCLabelBMFont类

CCLabelBMFont类支持不同宽度的字符，并且有很多第三方的编辑器支持，比CCLabelAtlas更灵活。

#### 认识CCLabelBMFont类

如果只需要使用标准的iOS字体，并且无需频繁修改标签的文本内容，那么使用标准的CCLabelTTF足矣。但有些时候在项目中需要使用定制的字体，而且需要每帧都更改文本的内容，此时就需要用到CCLabelBMFont。

在安装Cocos2D的目录中找到cocos2d-ios.xcodeproj，双击打开该项目。在Cocos2D文件夹下的Label Nodes中找到CCLabelBMFont.h，单击该文件名可以看到CCLabelBMFont类的定义，如图5-7所示。

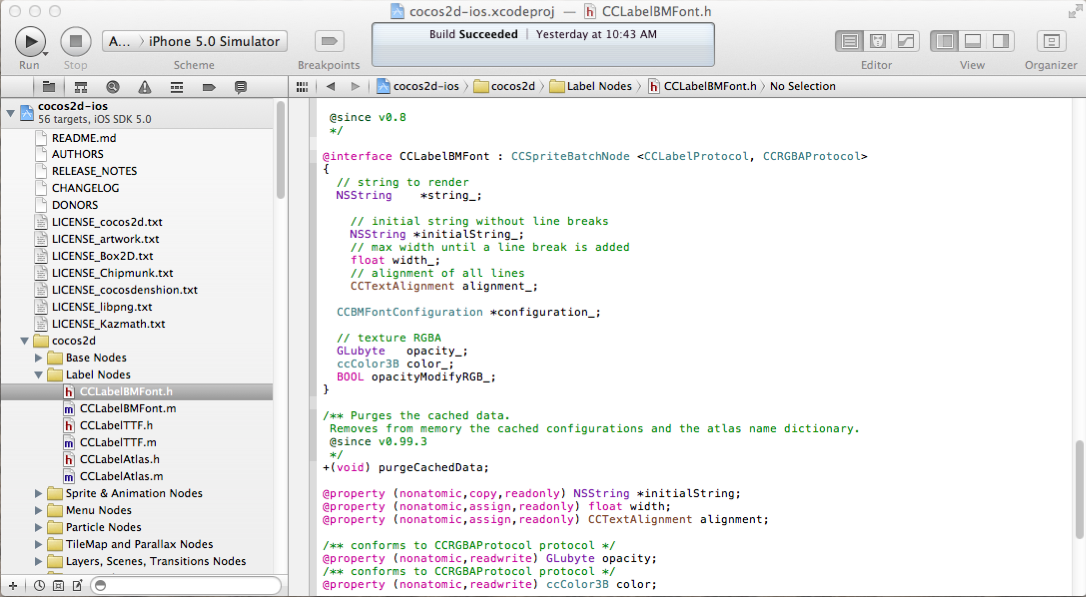


图5-7 在Xcode中查看CCLabelBMFont.h的定义

首先这个类的定义如代码清单5-7所示。

1. CCLabelBMFont类的定义

@interface CCLabelBMFont : CCSpriteBatchNode <CCLabelProtocol, CCRGBAProtocol>

{

// string to render

NSString \*string\_;

CCBMFontConfiguration \*configuration\_;

// texture RGBA

GLubyte opacity\_;

ccColor3B color\_;

BOOL opacityModifyRGB\_;

}

/\*\* Purges the cached data.

Removes from memory the cached configurations and the atlas name dictionary.

@since v0.99.3

\*/

+(void) purgeCachedData;

/\*\* conforms to CCRGBAProtocol protocol \*/

@property (nonatomic,readwrite) GLubyte opacity;

/\*\* conforms to CCRGBAProtocol protocol \*/

@property (nonatomic,readwrite) ccColor3B color;

/\*\* creates a BMFont label with an initial string and the FNT file \*/

+(id) labelWithString:(NSString\*)string fntFile:(NSString\*)fntFile;

/\*\* init a BMFont label with an initial string and the FNT file \*/

-(id) initWithString:(NSString\*)string fntFile:(NSString\*)fntFile;

/\*\* updates the font chars based on the string to render \*/

-(void) createFontChars;

@end

/\*\* Free function that parses a FNT file a place it on the cache

\*/

CCBMFontConfiguration \* FNTConfigLoadFile( NSString \*file );

/\*\* Purges the FNT config cache

\*/

void FNTConfigRemoveCache( void );

从上面的代码中可以看到，CCLabelBMFont类直接继承自CCSpriteBatchNode，并实现了两个协议，分别是CCLabelProtocol和CCRGBAProtocol。

CCLabelBMFont的特有属性如下：

* **opacity**：透明度，变量类型是Glubyte。
* **color**：色彩，变量类型是ccColor3B（使用RGB三种色彩通道数值表达的色彩）。

CCLabelBMFont类提供一个用于初始化的类方法和实例方法。

因为CCLabelBMFont继承自CCSpriteBatchNode，所以它将文本中的每个字符作为一个独立的CCSprite精灵对象看待。因此每个字符的旋转角度、大小、着色和透明度等属性都可以被修改。此外，第3章提到过，在CCMenuItemLabel中可以内置Label对象，因此CCLabelBMFont可以当作菜单项的一部分。同时，和CCLabelTTF一样，可以更改锚点来设置标签的对齐方式。

提示 CCLabelBMFont具备了CCLabelTTF所缺乏的灵活性，同时又具备CCLabelAtlas的高速渲染速度，以及CCSprite的所有特性。如果不清楚在项目中该使用哪种标签，那么CCLabelBMFont无疑是首选。

#### 如何使用CCLabelBMFont

在了解了CCLabelBMFont的理论知识，以及如何使用Hiero来创建字符图集，接下来将使用一个实例来说明如何使用CCLabelBMFont类。

1. 在Xcode中创建一个新项目。

打开Xcode，单击File→New，选择New Project，在iOS下选择Cocos2D，选择Cocos2D标准模板，单击Next，在Product Name处填上LabelBMFont，选择存储的位置，单击Create。

1. 加载资源文件。

把myfont.fnt和myfont.png文件（存在chapter5/resource/fonts/目录中）拖到Xcode的Resources中，确保选中“Copy items into destination group’s folder(if needed)”。

1. 修改init方法。

切换到HelloWorldLayer.m文件，找到init方法，更改其中创建标签的代码，修改后如代码清单5-8所示。

1. init方法的实现代码

-(id) init

{

// always call "super" init

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

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

// 使用CCLabelBMFont的类方法创建了一个标签，并使用刚刚在Hiero中生成的myfont.fnt字符图集

CCLabelBMFont \*label = [CCLabelBMFont labelWithString:@"Hello " fntFile:@"myfont.fnt"];

// 获取屏幕的大小

CGSize size = [[CCDirector sharedDirector] winSize];

// 将标签放置在屏幕的中心位置

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

// 将标签添加为当前层的子节点

[self addChild: label];

}

return self;

}

编译运行项目，会在屏幕上看到绿色的Hello标签。如图5-11所示。



图5-11使用CCLabelBMFont创建的标签

此时同样可以给这个标签一些动作。需要记住，只要是CCNode节点类或其子节点类，都继承它的各个属性和方法，包括动作。而CCLabelBMFont也属于节点类，自然也不例外。

现在让绿色的Hello标签在2秒内放大到原来的4倍，同时淡入。方法：在HelloWorldLayer.m中找到init方法，找到将标签添加到层的子节点这句代码，后面添加以下代码：

id labelAction = [CCSpawn actions:[CCScaleBy actionWithDuration:2.0f scale:4],[CCFadeIn actionWithDuration:2.0f], nil];

[label runAction:labelAction];

提示 细心的读者会发现，当标签放大到原来的4倍时，边缘会出现一些毛刺，而这一现象在使用CCLabelTTF时是不会存在的。因为CCLabelBMFont是纹理图，类似位图，而CCLabelTTF类似矢量图。大家都知道位图在放大时会出现模糊或失真现象，而矢量图则不会。所以，在使用CCLabelBMFont时，不能把缩放比例设置过大。通常情况下，对于同一字符的不同大小，需要设置不同的字符图集。

## 标签的对齐方式

第3章介绍过CCNode的一个基本属性锚点（anchor point）。所有CCNode节点，包括CCLabelTTF都有锚点属性，用来明确节点在屏幕中的位置。对于标签来说，锚点的设置非常有用。

通常无需对锚点值做任何调整，但如果想设置标签的不同对齐方式，使用锚点是不错的选择。示例代码如代码清单5-9所示。

1. 尝试设置不同的锚点值

//尝试设置不同的锚点值

CCLabelTTF \*label4 = [CCLabelTTF labelWithString:@"对齐方式" fontName:@"Cochin" fontSize:20];

label4.position = ccp(size.width/2,-120+size.height/2);

//右对齐

// label4.anchorPoint = ccp(0,0.5f);

//左对齐

// label4.anchorPoint = ccp(1,0.5f);

//顶部对齐

// label4.anchorPoint = ccp(0.5f,0);

//底部对齐

// label4.anchorPoint = ccp(0.5f,1.0f);

//默认的几何中心位置

label4.anchorPoint = ccp(0.5f,0.5f);

// 将标签对象添加为当前层的子节点

[self addChild:label4];

注意 如果修改锚点的位置，在旋转节点对象时也会以锚点位置为基础。

## 使用第三方工具创建字体纹理图集

位图字体图集其实是一张图片，其中包含了所有要显示的字符，及其描述字符在字体图集中位置的坐标数据。第3章中曾经接触过CCSpriteBatchNode和纹理图集，既然CCLabelBMFont是CCSpriteBatchNode的子类，那么它的作用就相当于精灵表单，而其中每个字符则相当于精灵表单中的单个精灵对象。

字符是以图片的形式存储，因此无法直接修改其大小，必须使用另一种字体图集。如果在游戏中需要使用两种及以上大小的相同字符，必须为每一种大小的字符创建单独的字体图集。

Cocos2D中使用FNT文件格式保存位图字体。虽然Cocos2D本身没有提供创建字体图集的工具（也没有提供创建纹理图集的工具），但可以使用第三方工具来完成这个工作。

Cocos2D官方推荐了以下几款字体图集创建工具：

1. Glyph Designer

商业软件，适用于Mac系统，其下载地址:http://glyphdesigner.71squared.com/。

1. Hiero

使用Java开发的免费软件，其下载地址:http://www.n4te.com/hiero/hiero.jnlp。

1. Slick2D Hiero Bitmap Font Generator

使用Java开发的免费软件，下载地址:<http://slick.cokeandcode.com/demos/hiero.jnlp>。

这里要介绍的是其中第二款软件Hiero，它是一款免费的字体图集创建工具，其作用类似于用于创建纹理图集的Zwoptex。

### Hiero工具的使用方法

Hiero（Hiero Font Builder Tool）是基于浏览器的免费Java应用，可以轻松创建用于创建Cocos2D的字体纹理图集。

打开浏览器，在地址栏中输入“www.n4te.com/heiro/hiero.jnlp”。

注意 当输入以上地址时，Safari浏览器的默认行为只是下载Hiero.jnpl文件。所以需要在

Downloads窗口中双击该文件以启动Hiero工具。

打开Hiero后，在界面左上的文本框中可以看到系统的可用字体，如图5-8所示。

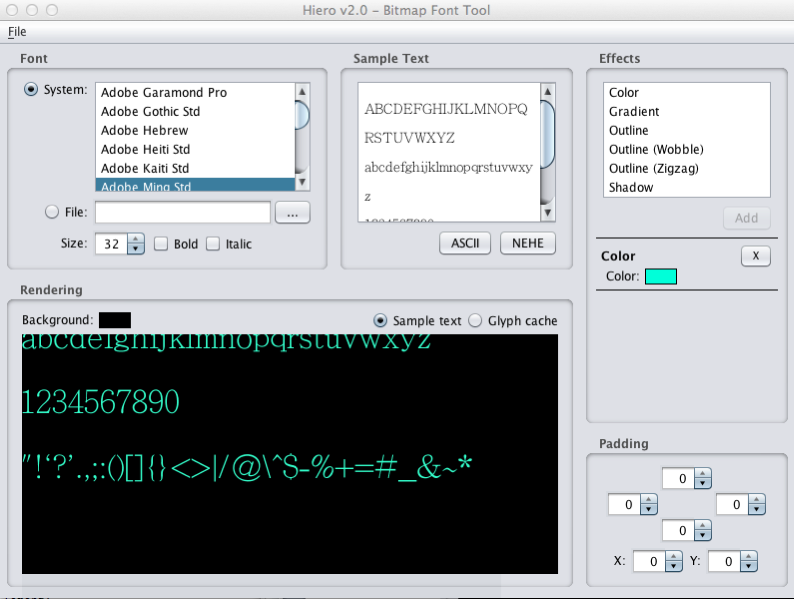


图5-8 Hiero主界面

### 创建字体图集步骤

这里示范如何使用Hiero创建一个字体图集，步骤如下：

* 1. 在界面左上的文本框中下滚，选择自己喜欢的字体，如AppleGothic。
  2. 在Sample Text部分，单击下方两个按钮（ASCII和NEHE）中的“NEHE”按钮，从而输入一些样本字符。如果只需要几个字符，也可以在那里手动输入。
  3. 在右边中间的Color下面，单击Color旁边的选色框，从弹出的窗口中选择自己喜欢的颜色，如图5-9所示。

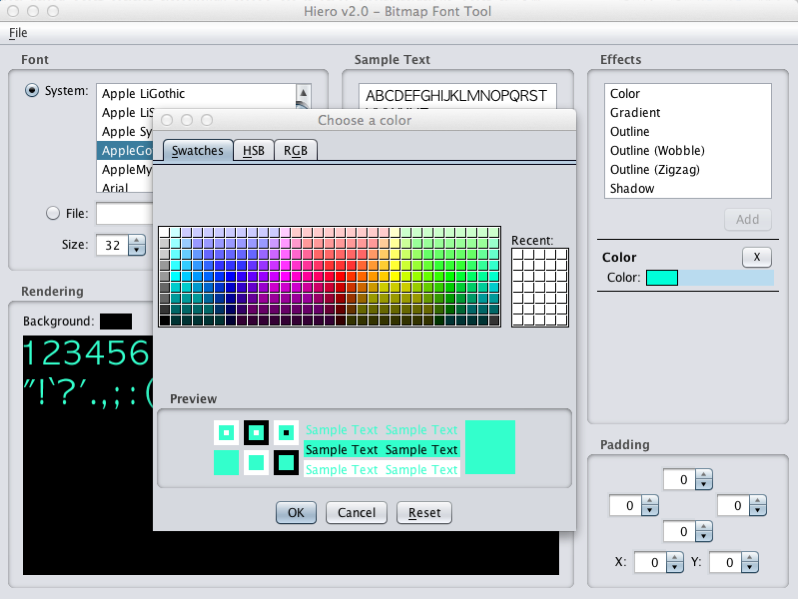


图5-9在Hiero中设置字体色彩

* 1. 在系统字体下方的Rendering部分，切换到Glyph cache，将Page width和Page height设置为256。

使用纹理图集时，希望用最小的2的乘方大小图像包含所有需要的字体。此处的设置表明，Hiero将创建一个256\*256的纹理图集，其中包含样本文本的所有字符，如图5-10所示。

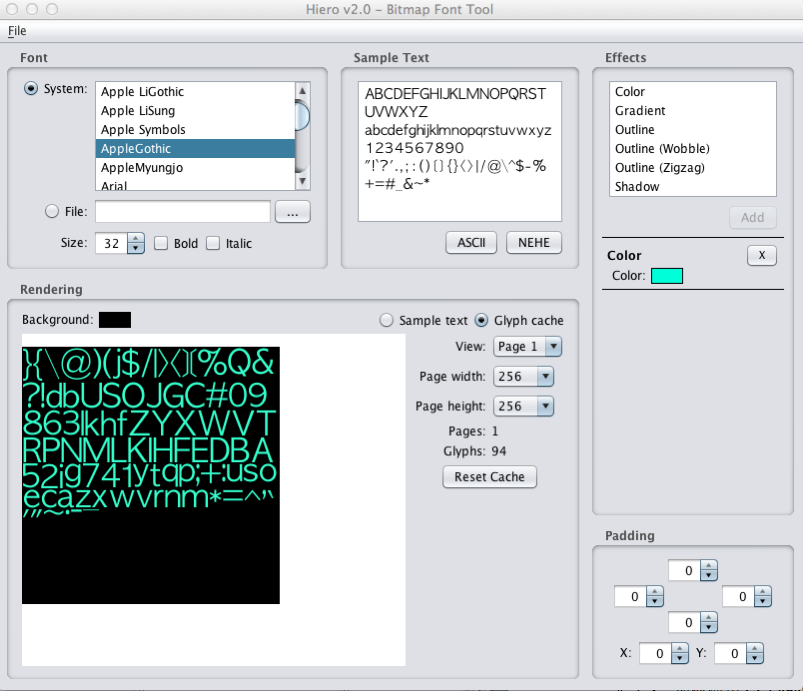


图5-10在Hiero中设置纹理图集大小

* 1. 单击File，选择Save BMFont Files，将文件命名为myfont.fnt。

Hiero将同时创建一个FNT文件和一个PNG文件（类似Zwoptex同时创建一个PLIST文件和一个PNG文件）。

注意 在游戏中需要显示的所有字符都需要在字符图集中呈现。如果使用了小写或大小字符，那么两种字符都应包含在字符图集中。

## 垂直射击游戏：美化游戏界面

前面几章使用的都是CCLabelTTF字体，为了美化游戏界面，本节进行重构，把游戏过程中动态改变的CCLabelTTF全部改用CCLabelBMFont类（主要是\_lifeLabel和\_scoreLabel）。此外，添加一个计时器，显示游戏当前剩余时间（精确到微秒当然得用CCLabelBMFont实现啦）。

首先，打开随书自带的模板。该模板在第4章代码的基本上添加一个位图字体文件num.fnt（位于chapter5/resource/fonts/目录下），及其对应的纹理图片num.png。打开Xcode项目，文件布局如图5-12所示。

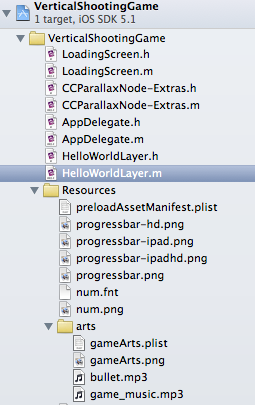


图5-12 Xcode项目的文件布局

渲染文本

把\_lifeLabel和\_scoreLabel换成CCLabelBMFont。

打开HelloWorldLayer.h，修改这两个label的定义如代码清单5-10所示。

1. 修改两个label的定义

CCLabelBMFont \*\_lifeLabel;

CCLabelBMFont \*\_scoreLabel;

打开HelloWorldLayer.m，修改init方法中的代码片断12，如代码清单5-11所示。

1. init方法中的代码片断修改

\_lifeLabel = [CCLabelBMFont labelWithString:@"0" fntFile:@"num.fnt"];

lifeLabel.position = ccpAdd(lifeIndicator.position, ccp(lifeIndicator.contentSize.width+10,0));

[self addChild:\_lifeLabel z:10];

\_scoreLabel = [CCLabelBMFont labelWithString:@"00" fntFile:@"num.fnt"];

\_scoreLabel.position = ccpAdd(scoreIndicator.position, ccp(scoreIndicator.contentSize.width+ 10,0));

[self addChild:\_scoreLabel z:10];

编译并运行，效果如图5-13所示。虽然看不出明显的变化，但是文本渲染性能得到了大幅提升。



图5-13文本渲染效果

提示 使用CCLabelBMFont显示分数等游戏中动态文本是最佳实践。

实现倒计时功能

添加游戏倒计时的功能步骤如下。

1. 打开HelloWorldLayer.h，添加新的实例变量CCLabelBMFont \*\_countdownLabel。
2. 找到HelloWorldLayer.m文件中的init方法，在init方法最后添加代码如代码清单5-12所示。
3. 在init方法的最后添加代码

//17.init consumedTime

\_totalSeconds = 60;

\_consumedTime = 0.0f;

NSString \*totoalSecStr = [NSString stringWithFormat:@"%d",\_totalSeconds];

\_countdownLabel = [CCLabelBMFont labelWithString:totoalSecStr fntFile:@"num.fnt"];

\_countdownLabel.position = ccp(winSize.width / 2, winSize.height - 50);

[self addChild:\_countdownLabel z:1000];

这段代码主要是初始化一个倒计时的标签。

1. 在update函数里更新些标签的值。代码如代码清单5-13所示。
2. 在update函数里更新标签值代码

-(void) update:(ccTime)dt{

if (!\_isGameStarted) {

return;

}

\_consumedTime += dt;

if (\_consumedTime >= \_totalSeconds) {

//game over

[self gameOver];

}

float result = \_totalSeconds-\_consumedTime;

int sec = result \* 100 / 100;

int microsec = (int)(result \* 100) % 100;

NSString \*str = [NSString stringWithFormat:@"%2d %2d",sec,microsec];

[\_countdownLabel setString:str];

。。。（下面的代码已省略）

}

上面的代码主要根据游戏总时间和当前消耗的时间计算label的显示数值。

1. 为了使倒计时的功能更加酷一些，打开init方法，在最后添加代码清单5-14所示代码。
2. init方法最后添加的代码

CCSprite \*char1 = (CCSprite\*)[\_countdownLabel getChildByTag:0];

CCSprite \*char4 = (CCSprite\*)[\_countdownLabel getChildByTag:3];

id scaleTo = [CCScaleBy actionWithDuration:1.0 scale:1.5];

id scaleBack= [scaleTo reverse];

id seq = [CCSequence actions:scaleTo,scaleBack, nil];

id ac = [CCRepeatForever actionWithAction:seq];

[char1 runAction:ac];

[char4 runAction:[[ac copy] autorelease]];

这里取出标签中的第1个和第4个字符，使用CCLabelBMFont的getChildByTag取出对应位置的字符。

需要注意两点：

* 每个字符都是一个CCSprite，可以对其进行任何关于CCSprite的操作。
* 在每个字符串初始化时，根据它们在字符串中的位置，下标从0开始。逐个添加相应位置的tag值。在本例中getChildByTag：0指的是获取第一个字符。

编译并运行，效果如图5-14所示。



图5-14 添加倒计时文本

## 本章小结

本章主要介绍Cocos2D的文本渲染系统，包括用于渲染文本的三个重要类CCLabelTTF、CCLabelAtlas和CCLabelBMFont，详细分析了这三个类的特性及其区别，并且使用三个示例说明每个类的具体使用方法；此外，学习如何使用Hiero字体创建工具创建字符图集。

在第6章将了解Cocos2D的事件处理机制，即如何和用户进行交互的问题。