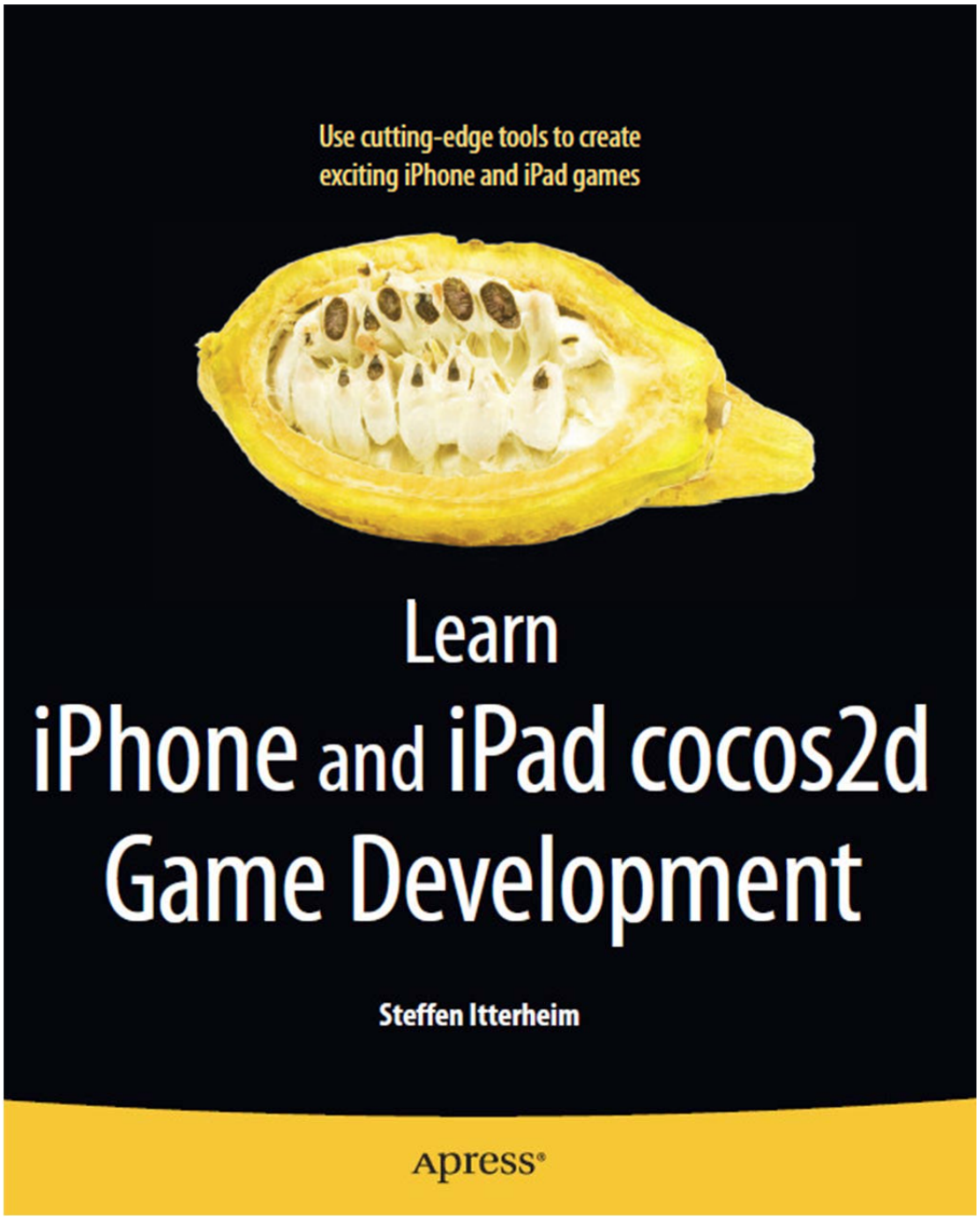

译者：杨栋
邮箱：yangdongmy@gmail.com

第十章
瓷砖地图（Tilemap）基础知识

在接下去的两章里，我将介绍基于瓷砖地图的游戏。不管你是不是从经典的角色扮演类游戏(例如Ultima）就已经开始玩游戏了，还是最近才开始在Facebook上玩Farmville，我敢肯定你已经玩过利用瓷砖地图来显示图形的游戏。

在瓷砖地图游戏里，游戏图形由叫做“瓷砖”（tiles）的一小组图片相互排列组成。这些图片被放置在一个网格中，得到的效果就是令人信服的游戏世界。瓷砖地图的概念非常吸引人，因为你可以节省内存而不必使用很多贴图渲染整个世界，同时还可以有很多不同的组合。

本章将会使用最简单的一种瓷砖地图：90度角瓷砖地图（Orthogonal Tilemaps），介绍瓷砖地图的一般概念。它们是用正方形或长方形的瓷砖组成的，通常以从上到下的视角展示游戏世界。例如，Ultima系列就已经使用瓷砖地图很长时间了。Ultima 1 到 Ultima 5 使用了正方形的瓷砖和从上到下的视角；Ultima 6和 Ultima 7 转换到“半斜45度角”(semi isometric)视角，但是仍然使用90度角瓷砖地图。Ultima 8:Pagan是这一系列中唯一使用“斜45度角”（isometric）瓷砖地图的游戏，所生成的地图创造出了更加令人身临其境的游戏世界。我们将在下一章讨论“斜45度角”瓷砖地图。

我将会在本章解释如何移动瓷砖地图，如何将地图定位在某块指定的瓷砖上，还有如何将屏幕保持在瓷砖地图的可视区域。瓷砖地图的移动是通过触摸瓷砖来实现的，这意味着你也会学习如何判断哪块瓷砖已经被触摸了。

什么是瓷砖地图（Tilemap）？

瓷砖地图是由多个单独的瓷砖组成的2D游戏世界。你可以利用几种拥有相同尺寸的图片创造出很大的游戏地图。这意味着瓷砖地图可以为大地图节省很多内存空间，它们首先出现在早期的电脑游戏中也就不足为奇了。很多经典的角色扮演类游戏使用正方形的瓷砖创造出了不可思议的幻想世界，有些看上去有点像**图10-1**中展示的瓷砖地图。

通常我们使用编辑器来编辑瓷砖地图。cocos2d直接支持的编辑器叫做Tiled(QT) Map Editor。Tiled是一款免费的开源工具，你可以用它编辑90度角瓷砖地图和斜45度角瓷砖地图，支持多个层。Tiled还允许你添加触发区域和物体，也可以为瓷砖添加代码中所需的，用来判断瓷砖类型的属性。

注:Qt是指诺基亚的Qt Framework，Tiled就是用Qt编写的。因为还有一个Java版本的Tiled，所以用Tiled(Qt)可以和Java版本很清楚地分开。Java版本已经停止更新，不过Java版本中有几个额外的功能是目前Qt版本中所没有的，所以值得一试。不过在本章和接下去的一章中，我将使用Tiled(Qt)来做演示和讨论。

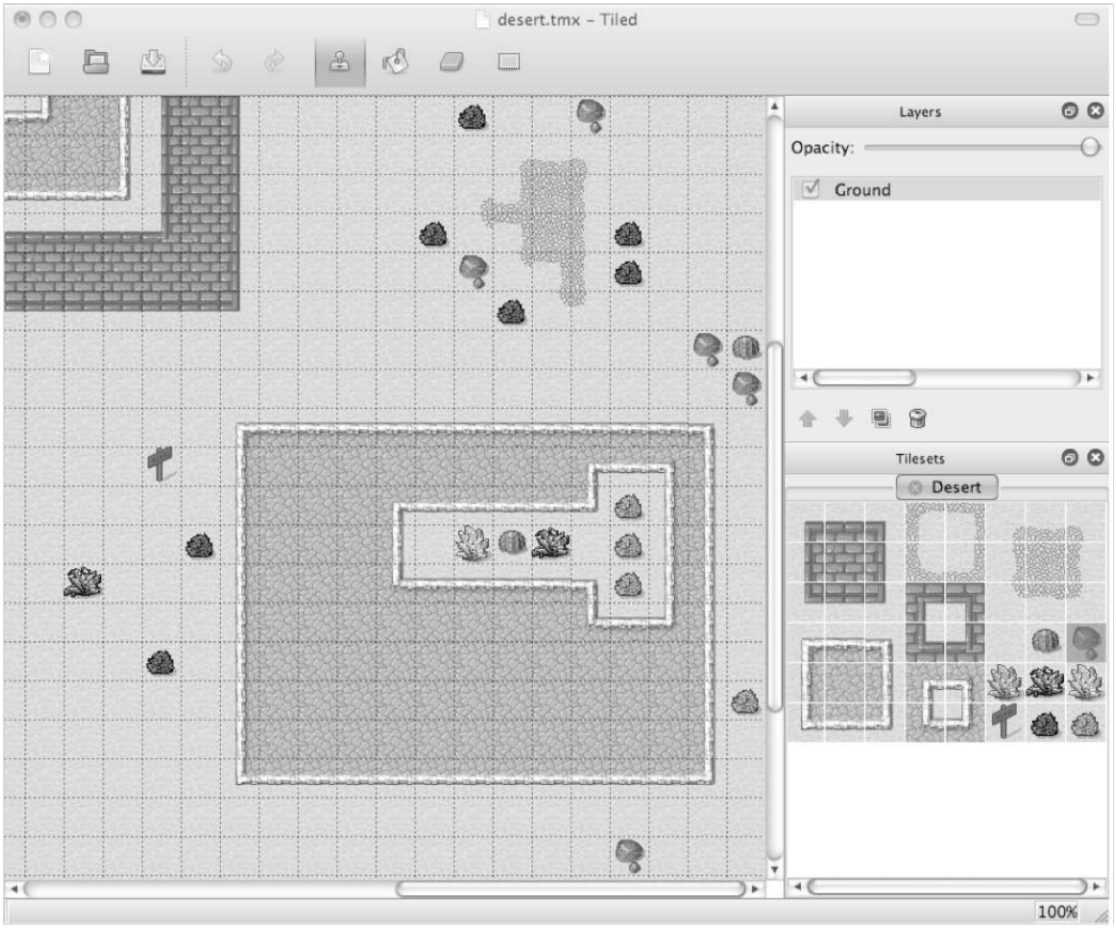

图10-1. 在Tile(Qt) Map Editor中的90度角瓷砖地图（Orthogonal Tilemap）

久而久之，人们开始在正方形瓷砖地图中添加“过渡瓷砖”（Trasition Tiles）。例如，我们不再直接在青草瓷砖旁边放置水的瓷砖，而是添加混合瓷砖（在这个例子中就是青草和水的混合瓷砖，青草在一边，水则在另一边，两者之间是一条分隔线），从而在水与草之间生成非常平滑的过渡。如果没有这个功能，你将不得不制作很多瓷砖，然后小心地考虑什么瓷砖可以过渡到其它瓷砖上去。

图10-1中的瓷砖地图有许多个很好的过渡瓷砖。沙漠瓷砖集只有4种地板瓷砖：沙漠，碎石（在瓷砖地图的下半部份），砖石（在左上角区域）和泥土（在右上角区域）。对于其中的3种瓷砖（除了沙漠），每一种都有12个瓷砖可被用于过渡到沙漠瓷砖。

瓷砖不一定是要正方形的；你也可以用长方形图片生成90度角瓷砖地图。亚洲的角色扮演类游戏通常使用长方形图片，例如Dragon Quest4到6。在使用90度角透视的同时，设计师可以使用长方形图片创造出长度比宽度大的物体，由此创造出深度的幻觉。

斜45度角瓷砖地图（Isometric Tilemaps）则通过将透视旋转45度以得到更加真实的深度感觉。通过制作3D风格的瓷砖，游戏世界获得了更多的视觉深度。虽然所有的瓷砖图片实际上是2D的，但是斜45度角瓷砖地图可以让我们的大脑

相信我们是在看3D的地图。斜45度角瓷砖地图的图片是钻石形状的（也就是菱形），同时允许靠近观察者的瓷砖覆盖离开观察者远一些的瓷砖。图10-2展示了一个斜45度角瓷砖地图。

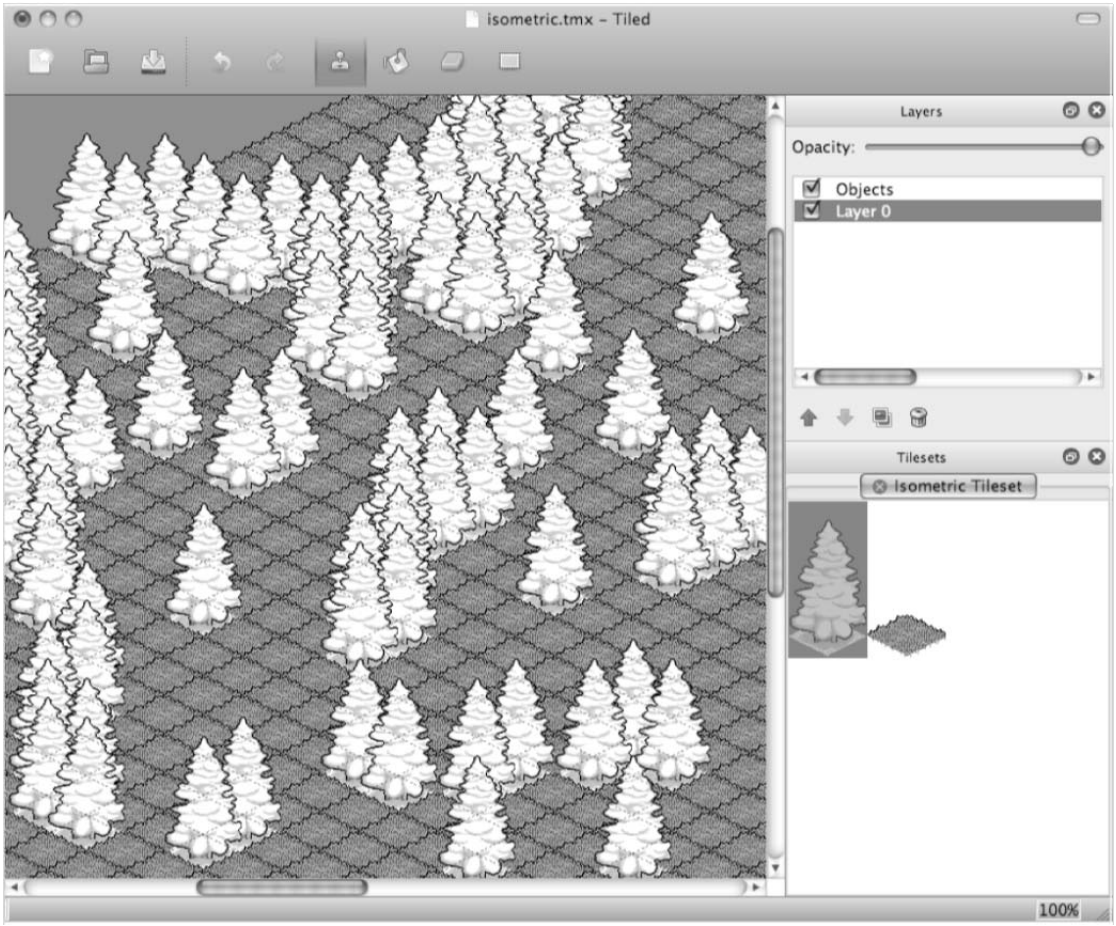

图10-2. Tiled(Qt) Map Editor中展示的斜45度角瓷砖地图

斜45度角地图证明了瓷砖地图不一定看上去是平的。如果将瓷砖设计成可以无缝地叠加在一起，你甚至可以用正方形的瓷砖地图得到和斜45度角瓷砖地图一样的效果。由于这个原因，Tiled支持使用多层瓷砖以生成令人信服的3D视图，如图10-3所示。多层瓷砖或者瓷砖的叠加也可用于斜45度角地图中，就像很多Farmville迷的视频展示的那样。有几个Farmville玩家只用了游戏中的玉米地就建造出了房子甚至高楼大厦。他们的秘诀是使用了斜45度角瓷砖地图中的视觉错觉。

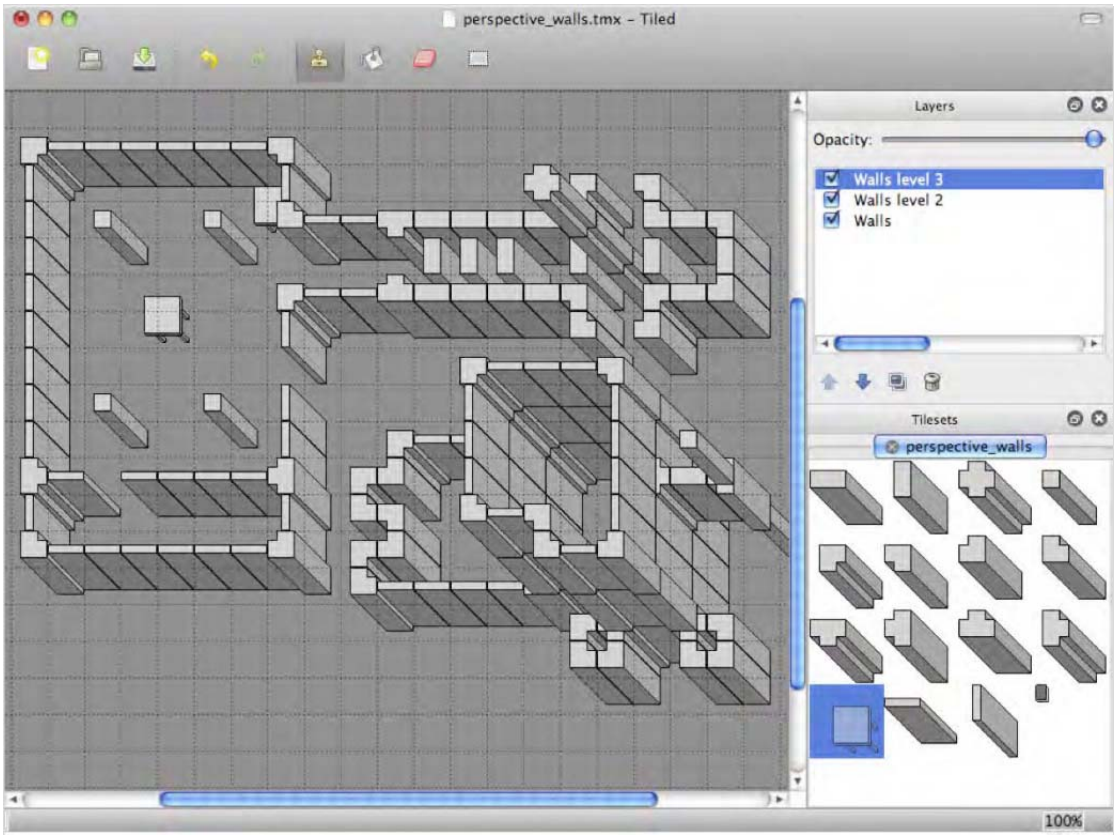

图10-3. 可以在多个层中使用的正方形瓷砖地图，用以创造出视觉深度。这种瓷砖地图是从游戏 Ultima 7 开始流行的。

在Zwoptex中准备图片

在本章的Tilmap01项目中，你会发现项目文件夹中的Resources/individual tile images文件夹里有几张正方形的瓷砖图片。将这些图片添加到Zwoptex中，然后将Canvas size设置为256x256像素-这个尺寸足够了。点击Apply按钮让Zwoptex自动排布这些图片。**图10-4**展示了排布的结果：

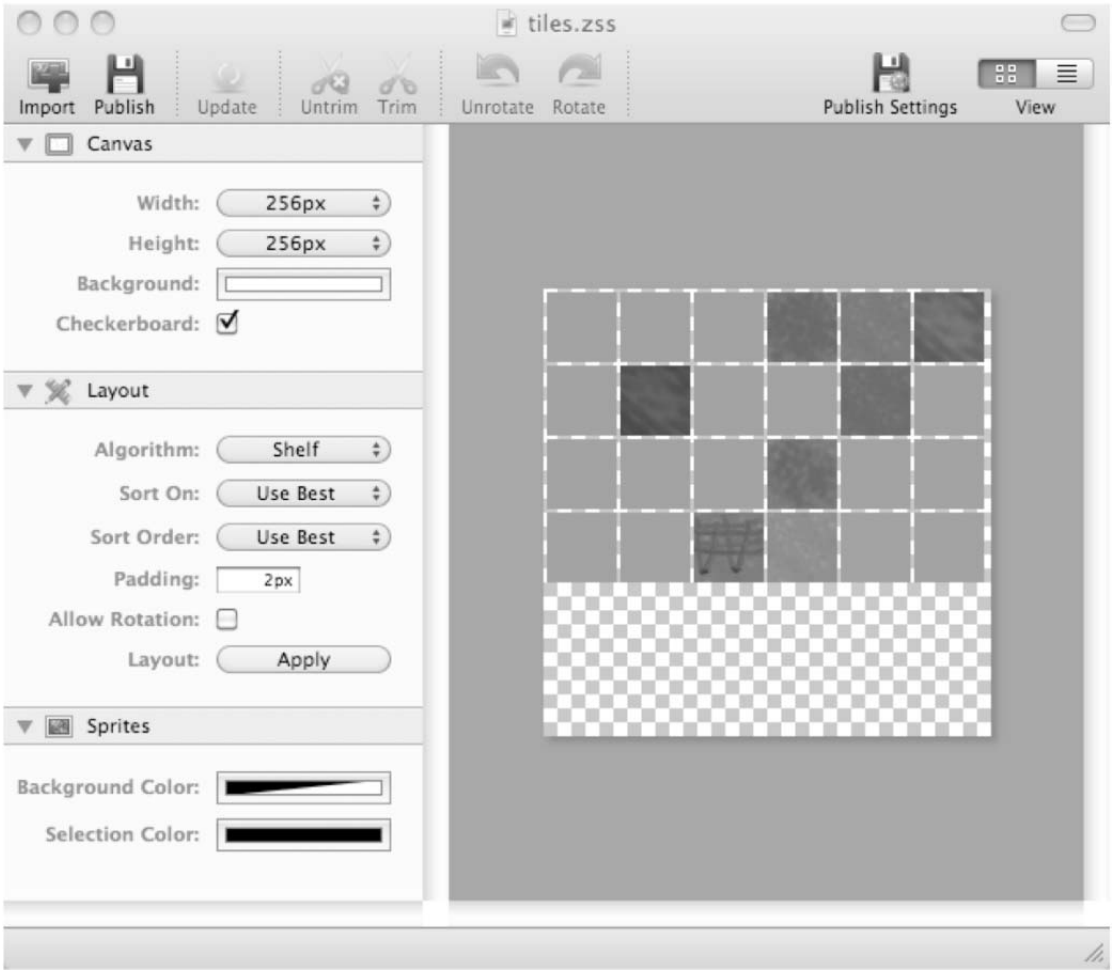

图10-4. 使用Zwoptex生成一张由几个正方形瓷砖图片组成的纹理贴图集

你会注意到Zwoptex是随机排布这些图片的。因为在撰写本书的时候，Zwoptex版本10.4还不能够按照瓷砖的名称来排布。如果可以用瓷砖名称排布的话，纹理贴图集的图片布局将会和硬盘上的文件名排布相同。已经有一些Zwoptex的用户请求了这个功能，所以在将来版本中应该会有以图片名称排序的布局方式。看一下是否你的Zwoptex版本支持按名称排序，如果有这个功能，你可以编辑瓷砖的命名，然后用这些图片生成一个以文件名排布的瓷砖集。

目前为止，你只能使用这些随机排布的图片。不过，当你在画布上添加或者删除图片，再点击Apply按钮以后，之前的图片排列次序就不能保证了。Zwoptex会再次随机排布图片的位置。不过，如果你是在CCSpriteBatchNode中通过引用图片名称来使用这些图片的话，上述随机排列并不会给你带来任何麻烦。

但是，对于Tiled Map Editor来说，瓷砖图片保持在相同的位置是很重要的，因为编辑器是通过图片位置和相关的位移来引用各个瓷砖的。这意味着如果贴图集里的瓷砖改变了位置，编辑器中的瓷砖地图将会看上去和之前的完全不一样。因为编辑器还在引用之前的瓷砖位置信息，但是之前可能是草地的地方现在可能已经成了水的瓷砖。

为了解决上述问题，你可以使用一些空白的瓷砖图片填充一定尺寸的纹理贴图集。空白图片的数量至少要和需要的瓷砖图片数量相同，或者更多一些。使用

空白图片的目的是为你创造可以手动描绘瓷砖的空间。你在Zwoptex中导入所有的空白图片，用于生成一个空白瓷砖平均分布的纹理贴图集。然后你可以关闭Zeoptex，你不再需要它了，因为你可以把得到的纹理贴图集在任何图片编辑器中打开，手动地将需要的瓷砖图片填充到非透明的空白区域。Zwoptex在这里的作用是帮助一开始的瓷砖排布。

如果你会画画，你也可以考虑直接在图片编辑软件中制作瓷砖地图。你需要注意的是要把图片的背景设置为透明。这样的话，当瓷砖显示在游戏里的时候，可以避免瓷砖边缘产生明显的毛边。还有，所有的瓷砖都要是相同尺寸的，而且瓷砖之间的空间必须保持一致。

对于我来说，我会将空白瓷砖图片导入Zwoptex，让它自动为我排布图片。你只需要做一次就够了。

Tiled Map Editor（瓷砖地图编辑器）

最出名的用于生成cocos2d中可用的瓷砖地图的编辑器叫做 Tiled Map Editor（在此我简称它为 Tiled）。cocos2d游戏引擎原生支持 Tiled 生成的TMX文件。Tiled是免费的，在撰写本书时，它的最新版本是0.5。你可以通过以下网址下载：http://www.mapeditor.org

如果你想支持Tiled进一步的开发工作，可以考虑向此软件的开发者 Thorbjørn Lindeijer 捐款。你可以通过以下网址捐款：
http://sourceforge.net/donate/index.php?group_id=161281.

生成一个新的瓷砖地图

在下载安装完成后，打开Tiled，点击菜单栏的View菜单，然后点击Tilesets和Layers两项。你会看到软件界面的右边会出现两块新的区域，一块在右边上半部份，显示层（Layers），另一块在右边下半部份，显示当前的瓷砖集（Tilesets）。瓷砖集其实就是一张包含多个瓷砖的图片，每个瓷砖之间的间隔相同，你也可以将其理解成一张仅包含相同尺寸图片的纹理贴图集。接着，从菜单栏选择 File ➤New，你会得到如图10-5所示的新地图生成对话框：

图10-5. Tiled中的新瓷砖地图生成对话框

目前，Tiled 支持90度角瓷砖地图和斜45度角瓷砖地图。

地图的尺寸是由瓷砖的数量来决定的，而不是像素。在我们的例子中，新地图的大小是 30x20块瓷砖，每块瓷砖的大小是32x32像素。单个瓷砖大小必须与单个瓷砖图片的大小相同，否则会导致瓷砖和图片大小不能完全匹配。

新生成的地图是空的，也没有任何瓷砖集加载进来以供使用。你可以通过使用菜单栏的 Map ➤ New Tileset 来添加需要的瓷砖集。点击New Tileset以后，你会得到如图10-6所示的New Tileset对话框，你可以在这里选择本地机器上的瓷砖集。

我在这里使用的瓷砖集是dg_grounds32.png。这些瓷砖图片是David E. Gervais通过“创作共享理念授权同意书”（Creative Common License）发布的，所以只要你在使用这些图片时说明图片是来自David E. Gervais的，你就可以免费分享和重新混合这些图片了。你可以通过以下网址下载更多David的瓷砖集：http://pousse.rapiere.free.fr/tome/index.htm

如图10-6所示，我用对话框中的Browse按钮在Tilemap01项目的Resources文件夹中找到了dg_grounds32.png瓷砖集图片。如果你勾选“Use transparent color”复选框，图片上的透明区域将被替换成粉色（默认颜色）。我没有勾选这个复选框，因为我们使用的图片中没有透明区域。

我们使用的瓷砖尺寸是32x32像素，这和瓷砖集图片中的单个瓷砖大小相同。Margin选项用于设置所有瓷砖离开整个大图边缘的距离；而Spacing则用于设置瓷砖之间的间隔距离。因为dg_grounds32.png中的单个瓷砖和我们使用的瓷砖大小相同，所有我将Margin和Spacing都设为0。

如果你使用Zwoptex生成瓷砖集图片的话，你必须将Zwoptex中的pixel-padding值填进Margin和Spacing这两个输入框中。默认情况下，Zwoptex使用的padding（填充）值是2个像素。

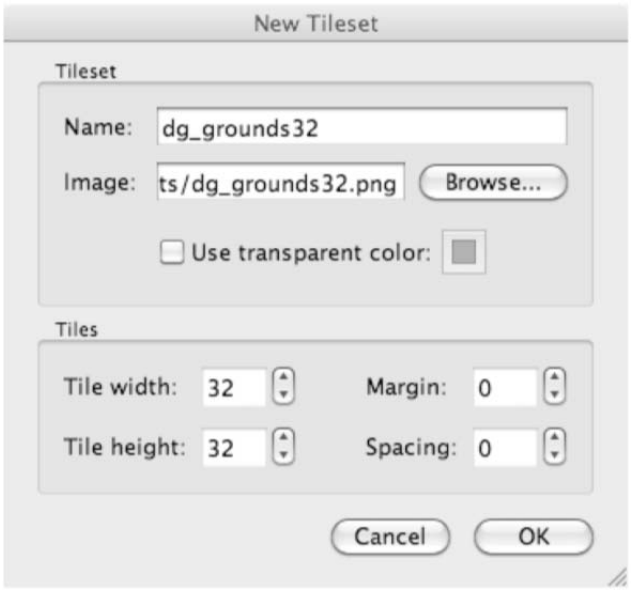

图10-6. 从外部加载图片生成新的瓷砖集（tileset）

当导入新的瓷砖集图片时，请确保将此图片导入你项目的资源文件夹。之后，当你导出 TMX 格式的瓷砖地图时，你要把这个TMX文件放到与瓷砖集图片相同的文件夹中。否则cocos2d会因为找不到TMX引用的瓷砖集图片而抛出程序运行异常。TMX文件只能引用同一文件夹下的瓷砖集图片。如果TMX文件和需要用到的瓷砖集图片在不同的文件夹中，当你的应用程序安装到模拟器或者真实设备上后，本来的文件夹结构就会发生变化，导致瓷砖集图片引用失败。

技巧：TMX文件实际上就是一个XML文件。如果你觉得好奇的话，你可以打开TMX文件查看。如果你看到里面的图片引用包含文件夹路径的话，cocos2d很可能会找不到相关的图片。图片引用的地方不应该包含任何路径，应该只有一个图片文件名，例如<image source=“tiles.png”/>。

设计瓷砖地图

完成瓷砖集图片的加载以后，在Tiled软件界面中，你看到的是一张空白地图。你可以在上面创建你的游戏世界。首先，我们要干掉这个空白的地图，给它一个默认的地板瓷砖。我选择了 Bucket Fill Tool，然后挑了一个明亮的草地瓷砖，让地图变成了一片郁郁葱葱的草地。图10-7展示了加载瓷砖集图片以后的软件界面。

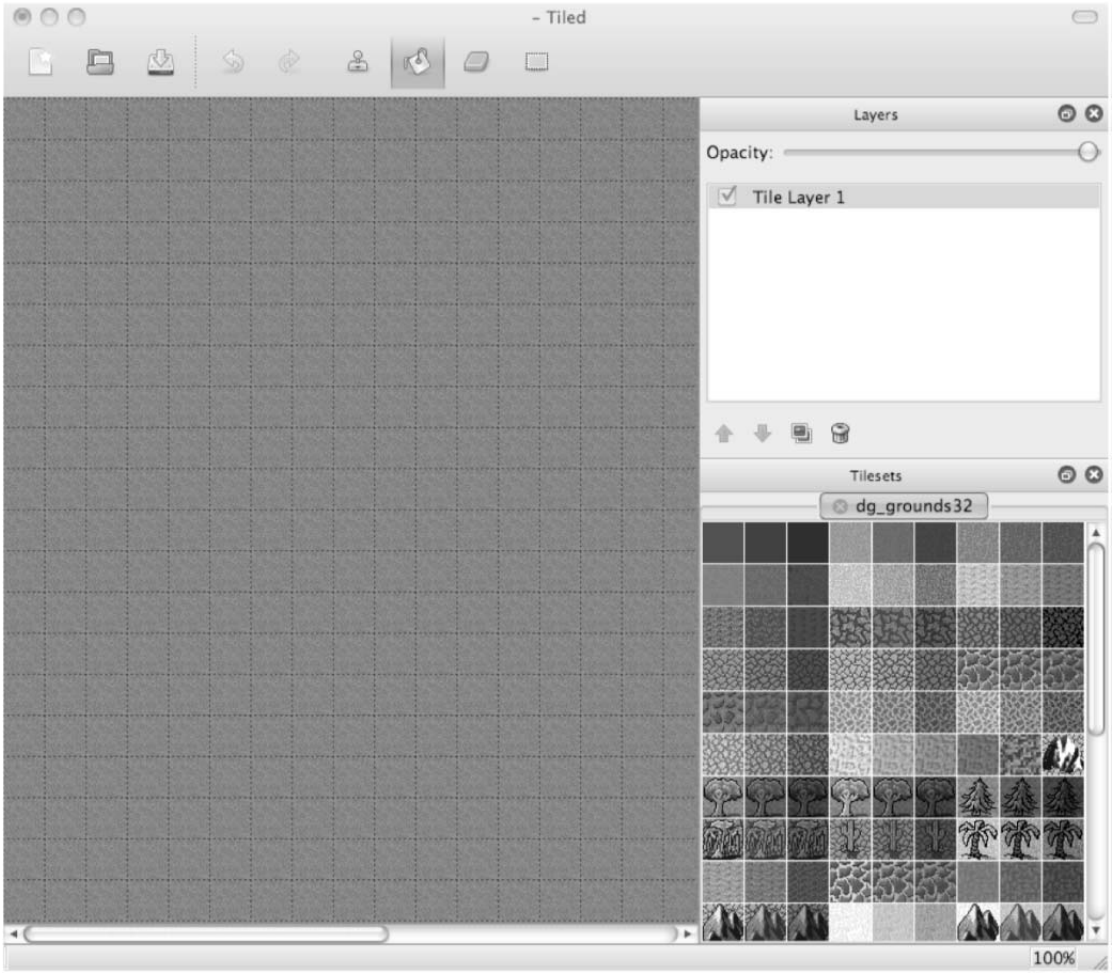

图10-7. 完成瓷砖集图片dg_grounds32.png加载以后的软件界面，呈现的是一个空白的地图。

Tiled有四种方式可以编辑瓷砖地图，菜单位于软件界面的右上角。它们分别是：Stamp Brush（快捷键 B），你可以把当前从瓷砖集中选择的瓷砖画在地图上；Bucket Fill（快捷键 F）,它可以填充空白区域或者由相同瓷砖连接起来的区域；Eraser（快捷键 E），用于擦掉瓷砖；Rectangular Select（快捷键 R），你可以选择一个区域，然后用Ctrl+c或者菜单项Edit > Copy来复制选中的区域，接着Ctrl+v或者菜单项Edit > Paste粘贴到当前鼠标在地图上所处的位置 - 在实际使用中你会看到，其实在粘贴时，软件是自动却换到了Stamp Brush模式进行粘贴的。还有个小技巧，如果你想清除当前的瓷砖选择，只要鼠标右键点击一下就可以了。

通常你会花很多时间使用Stamp Brush来选择不同的瓷砖，一块块地放在地图上，完成你基于瓷砖的游戏世界。

你也可以通过多个层来编辑瓷砖地图。你可以在右上方的Layers区域添加删除多个层。从顶部菜单选择 Layer > Add Tile Layer 可以生成一个新的瓷砖层。使用多个层的好处是你可以在cocos2d中将地图中的某个区域替换掉。在TileMap01项目中，我利用层把地图的一部份在冬天和夏天之间进行切换。

你可以通过 Layer > Add Object Layer 生成用于添加物体的层。Tiled中的物体只是简单的方块而已，你可以在里面进行绘画操作，之后则可以在代码中进行获取。你可以使用物体层触发某些事件 - 比如，当玩家角色进入一个区域时，系统会自动生成一些怪物。我在样例里随机添加了一些物体层，用于演示如何在cocos2d代码中使用它们。

Tiled的一些功能是隐藏在上下文菜单里的。例如，对于上述物体层中的长方形物体，你可以在物体区域内点击右键，在弹出菜单中选择Remove Object进行移除。注意，你必须将Layer视图中的物体层选中后，才能让右键的上下文菜单显示。

你也可以通过右键点击物体，层和瓷砖，在弹出菜单中选择它们的Properties菜单以编辑属性。修改属性的一个用处是：通过选择Layer ➤ Add Tile Layer，你可以生成一个瓷砖层。这个层将在游戏中被用于判断瓷砖的某些属性。我将这个新层命名为GameEventLayer（游戏事件层）。接着，选择Map ➤ New Tileset，然后在与dg_grounds32.png所处的同一文件夹中加载game-events.png。game-event.png只包含三种瓷砖。右键点击其中一块瓷砖，选择Tile Properties，然后如图10-8添加 isWater 属性。

注意：请记住，每一个瓷砖层都会带来额外的系统开销。如果你把瓷砖放在多个层中的相同位置，带来的开销就更大了。因为每个层都会被渲染，从而影响游戏运行性能。我的建议是层的数量越少越好。对大多数游戏而言，2到4个瓷砖层已经足够了。在添加了新层和设置完层中的瓷砖以后，你要确保在真实设备上做个测试，看一下游戏的帧率是否正常。

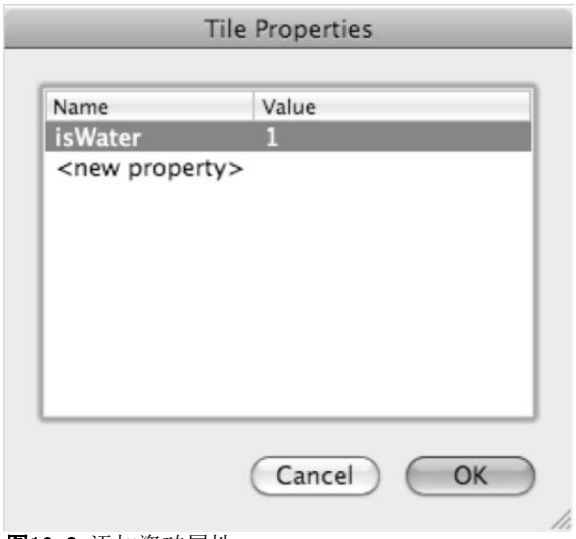

图10-8.添加瓷砖属性

然后你就可以用已经添加了isWater属性的瓷砖，在地图上放置这些瓷砖了。比如，你可以在河流上放置这种瓷砖－河流在下面一层。如果你想在放置瓷砖时看到底下一层的话，你可以在Layers视图中选择GameEventLayer，调节上面的Opacity滑块来调整层的透明度，或者也可以通过层名称前的复选框来显示或隐藏层。

请确保在保存TMX瓷砖地图前勾选所有层前面的复选框。cocos2d不会加载那些没有在Tiled里面勾选的层。

当你完成上述步骤以后，你应该会得到类似图10-9所示的瓷砖地图。将它保存在TileMap01项目的Resources文件夹中，里面也保存着瓷砖地图所引用的瓷砖集图片。

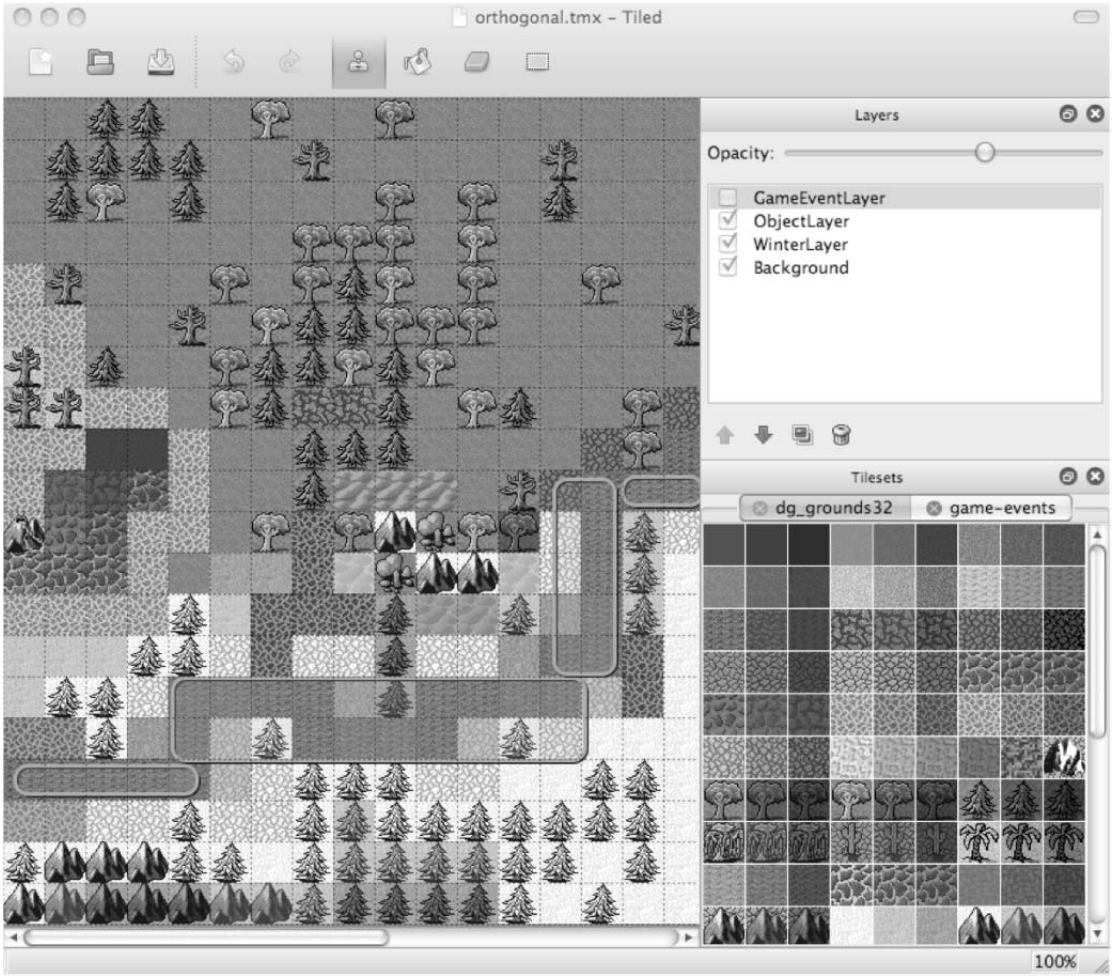

图10-9. 拥有三个瓷砖层和一个物体层的瓷砖地图

在cocos2d中使用90度角瓷砖地图

要在cocos2d中使用TMX格式的瓷砖地图，首先必须把TMX文件和它所引用的瓷砖集图片文件作为资源添加到Xcode项目中。在TileMap01项目中，我将orthogonal.tmx，dg_grounds32.png和gameevents.png作为资源添加到项目中。在代码里加载和显示瓷砖地图非常方便。以下代码来自TileMapLayer类的init方法：

```
CCTMXTiledMap* tileMap = [CCTMXTiledMap tiledMapWithTMXFile:@"orthogonal.tmx"];
[self addChild:tileMap z:-1 tag:TileMapNode];
CCTMXLayer* eventLayer = [tileMap layerNamed:@"GameEventLayer"];
eventLayer.visible = NO;
```

CCTMXTiledMap类是用TMX文件名来初始化的，然后作为子节点被添加到当前层中；TileMapNode这个tag可以允许用户以后再次访问tileMap节点。当然，你也可以用成员变量来储存tileMap节点。

下一步是通过使用tileMap的layerNamed方法和在Tiled中应用过的层命名“GameEventLayer”，来获取CCTMXLayer（也就是之前在Tiled中添加的游戏事件层）。因为游戏事件层只是cocos2d用来为某些瓷砖确定属性的，所以我们并

不需要渲染这个层。我通过eventLayer.visible = NO; 来隐藏这个层。不过请注意：如果你在Tiled软件中没有勾选游戏事件层前面的复选框的话，这个层不仅不会显示，你也不能使用层中包含的瓷砖属性信息。

如果你现在运行TileMap01项目，你将看到如图10-10所示的瓷砖地图。

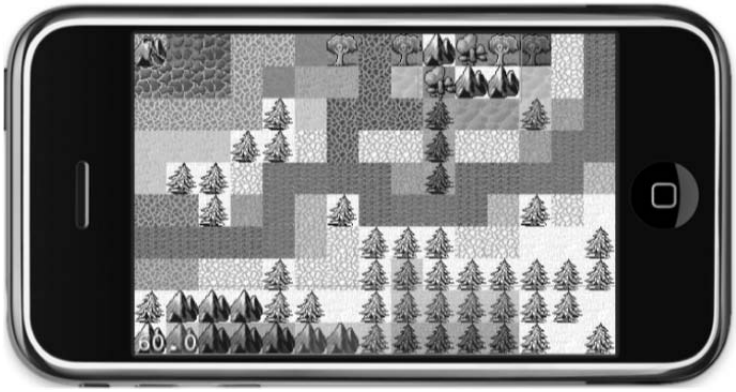

图10-10. iPhone模拟器中的90度角瓷砖地图

现在你还不能用瓷砖地图做任何事情。我们来做些改变。在TileMap02项目中，我想找到那些属性为isWater的瓷砖。我在类里添加了ccTouchesBegan方法用于检测玩家触摸了哪一块瓷砖，代码如列表10-1所示。

列表10-1. 检测瓷砖的属性

```
-(void) ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
	CCNode* node = [self getChildByTag:TileMapNode];
	NSAssert([node isKindOfClass:[CCTMXTiledMap class]], @"not a CCTMXTiledMap");
	CCTMXTiledMap* tileMap = (CCTMXTiledMap*)node;

	// 从触摸位置获取瓷砖的坐标信息
	CGPoint touchLocation = [self locationFromTouches:touches];
	CGPoint tilePos = [self tilePosFromLocation:touchLocation tileMap:tileMap];

	// 检查用户是否触摸在了水面上（就是说是否触摸了带 isWater 属性的瓷砖）
	bool isTouchOnWater = NO;

	CCTMXLayer* eventLayer = [tileMap layerNamed:@"GameEventLayer"];
	int tileGID = [eventLayer tileGIDAt:tilePos];

	if (tileGID != 0)
	{
		NSDictionary* properties = [tileMap propertiesForGID:tileGID];
		if (properties)
		{
			NSString* isWaterProperty = [properties valueForKey:@"isWater"];
			isTouchOnWater = ([isWaterProperty boolValue] == YES);
		}
```

```
        }

        // 取决于触摸的位置，决定下一步发生的动作
        if (isTouchOnWater)
        {
                [[SimpleAudioEngine sharedEngine] playEffect:@"alien-sfx.caf"];
        }
        else
        {
                // 获取冬季层，切换它的可视性属性
                CCTMXLayer* winterLayer = [tileMap layerNamed:@"WinterLayer"];
                winterLayer.visible = !winterLayer.visible;
        }
}
```

我通过tag得到了CCTMXTiledMap的节点。触摸的位置信息首先被转换成屏幕坐标，然后坐标被用于获取相对应的瓷砖地图上的坐标tilePos － 也就是触摸到的瓷砖位置信息。我将在稍后解释tilePosFromLocation方法。现在你只需要知道此方法会返回触摸到的瓷砖的位置信息。

现在我要给大家介绍一下瓷砖的“全局标识符”（GIDs）概念。瓷砖地图上的每一块瓷砖都附带一个独有的整数编号，也就是GID。地图上的瓷砖GID是从1开始连续编号的。GID为0表示的是空白瓷砖，也就是没有瓷砖。利用CCTMXLayer的tileGIDAt方法，你可以获取指定坐标上瓷砖的GID编号。

接下去的代码中，我从瓷砖地图上得到名为GameEventLayer的CCTMXLayer层。GameEventLayer层就是我放置了isWater瓷砖的层，这个层叠加在河流瓷砖层上面。tileGIDAt方法会返回触摸到的瓷砖GID编号。如果返回的GID编号为0，就说明层上的这个位置没有放瓷砖 － 也就说明我们触摸到的瓷砖不可能是带isWater属性的瓷砖。

CCTMXTiledMap有一个叫做propertiesForGID方法，如果指定GID编号的瓷砖附带属性的话，此方法会返回一个NSDictionary字典。这个NSDictionary字典包含了在Tiled中为瓷砖添加的属性（见图10-8）。字典的键/值都是用NSString对象存储的。如果你出于调试目的想看一下字典里到底有什么东西，你可以使用CCLOG将字典打印出来：

```
CCLOG(@"NSDictionary 'properties' contains:\n%@", properties);
```

上述代码会在调试窗口中打印与以下类似的信息：

```
2010-08-30 19:50:52.344 Tilemap[978:207] NSDictionary 'properties' contains:
{
        isWater = 1;
}
```

在使用瓷砖地图的时候，你将会处理好多种NSDictionary对象。用CCLOG将字典

内容打印出来可以帮助你了解你将要处理的字典内容。当然，你可以将任何iPhone SDK的集合类，比如说数组，用CCLOG打印出来。

NSDictionary的每一个属性都可以通过它的valueForKey方法来获取，返回的是一个NSString对象。要从一个NSString对象中获取布尔值，你可以使用NSString的boolValue方法；而要获取整数或者浮点数，你可以使用NSString的intValue或者floatValue方法。

在ccTouchesBegan方法的结束处，我检查了用户是否触摸了标记为水面的瓷砖，如果用户触摸了水面，游戏将会播放一段声音。否则，我会获取WinterLayer层，切换它的可视属性值。通过这样的方式，改变季节变得非常容易！这个效果也展示了你可以通过在Tiled设置多个层，来全局地改变瓷砖地图的显示，而不需要加载另一个独立的瓷砖地图。

如果你需要在地图上修改单个瓷砖，你可以使用removeTileAt和setTileGID这两个方法。在游戏过程中，前者会移除指定层上的瓷砖，后者则会替换指定层上瓷砖：

```
[winterLayer removeTileAt:tilePos];
[winterLayer setTileGID:tileGID at:tilePos];
```

定位触摸到的瓷砖

之前我提到过 tilePosFromLocation 这个方法，在这里我再重复一下之前的两行代码：

```
// 通过触摸坐标获取瓷砖在瓷砖地图上的坐标
CGPoint touchLocation = [self locationFromTouches:touches];
CGPoint tilePos = [self tilePosFromLocation:touchLocation tileMap:tileMap];
```

上述代码中，触摸的位置首先被转换成屏幕坐标。因为你经常需要用到这些代码，所以我在**列表10-2**中提供了这些代码：

列表10-2. 将用户在屏幕上的触摸输入坐标转换成屏幕坐标

```
-(CGPoint) locationFromTouch:(UITouch*)touch
{
        CGPoint touchLocation = [touch locationInView: [touch view]];
        return [[CCDirector sharedDirector] convertToGL:touchLocation];
}

-(CGPoint) locationFromTouches:(NSSet*)touches
{
        return [self locationFromTouch:[touches anyObject]];
}
```

触摸位置信息转换成屏幕坐标以后，我们调用了 tilePosFromLocation 方法。

此方法需要触摸位置（location）和tileMap的指针作为参数。列表10-3中的方法包含了一些数学，我将很快做出详细解释：

列表10-3. 将屏幕坐标转换为瓷砖在地图上的坐标

```
-(CGPoint) tilePosFromLocation:(CGPoint)location tileMap:(CCTMXTiledMap*)tileMap
{
        // 触摸的屏幕坐标必须减去瓷砖地图的坐标 — 万一瓷砖地图位置已经不在（0，0）点上了
        CGPoint pos = ccpSub(location, tileMap.position);

        // 将得到坐标值转换成整数
        pos.x = (int)(pos.x / tileMap.tileSize.width);
        pos.y = (int)((tileMap.mapSize.height * tileMap.tileSize.height - pos.y) / tileMap.tileSize.height);
        CCLOG(@"touch at (%.0f, %.0f) is at tileCoord (%i, %i)", location.x, location.y,
                                                                (int)pos.x, (int)pos.y);

        NSAssert(pos.x >= 0 && pos.y >= 0 && pos.x < tileMap.mapSize.width &&
                                        pos.y < tileMap.mapSize.height,
                                        @"%@: coordinates (%i, %i) out of bounds!",
                                        NSStringFromSelector(_cmd), (int)pos.x, (int)pos.y);
        return pos;
}
```

还和我的思路在一起吗？如果你以前用过瓷砖地图，你应该对上述代码挺熟悉的。如果你从来没有用过，那可能有些难以理解。上述方法做的第一件事情是用屏幕坐标减去tileMap.position（瓷砖地图的坐标）。在接下去的Tilemap03项目中，我加入了瓷砖地图的移动，所以瓷砖地图的位置很可能不会在（0，0）点上。

因为瓷砖地图的起始位置一开始是和屏幕的左下角对齐的，坐标是（0，0），所以瓷砖地图和屏幕的（0，0）点一开始是重合的。如果你想把瓷砖地图移到（100，100）像素的坐标点，屏幕上的效果看起来像是在向左下方移动。一个很普遍的误解是认为我们的视角在移动，实际上是瓷砖地图在移动，而且是向右上方移动的。因此，如果你要让屏幕视角向右上方移动，也就是向地图的中心点移动的话，你其实是要用触摸处的屏幕坐标减去负的瓷砖地图坐标：

location(240, 160) – tileMap.position(-100, -100) = pos(340, 260)

上面一行代码得到的坐标就是用户在屏幕上触摸到的瓷砖地图上的坐标。

现在我们得到了瓷砖在地图上的确切坐标值。接着我们要找到那块瓷砖。与屏幕坐标不同，每块瓷砖的坐标是从地图的左上角开始的，而不是屏幕坐标的左下角。图10-11展示了一系列瓷砖的x,y坐标。我使用的截图是通过使用Tiled的Java版中的 View ➤ Show Coordinates 功能得到的。Tiled的QT版本目前还没有这个功能。

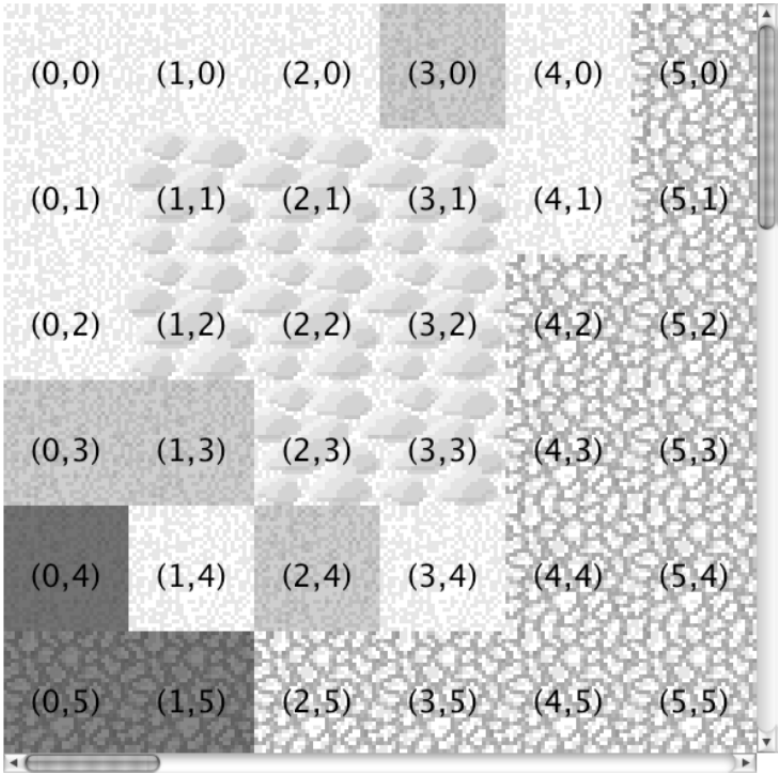

图10-11. 90度角瓷砖地图的坐标系统

以下代码用于计算瓷砖的x轴坐标：

```
pos.x = (int)(pos.x / tileMap.tileSize.width);
```

tileMap.tileSize属性是当个瓷砖的大小，在我们的例子里是32x32(请参考图10-6)。如果触摸的瓷砖地图x轴坐标是340像素的话，具体的计算将会是如下所示：

```
340 / 32 = 10.625
```

上述计算得到结果并不正确。因为我们需要的是瓷砖的x轴坐标，应该是一个整数才对！得到这个结果的原因是：上述触摸的区域是在一块瓷砖32x32正方形区域内。我们通过以下代码将浮点数转换成整数：

```
pos.x = (int)10.625  // pos.x == 10
```

使用(int)会把小数点后面部份移除。如果你移除小数点后的部份，直接用10.625作为坐标值来获取瓷砖的话，程序将会报错。因为没有一个瓷砖的x轴是在10.625上的，只有x轴在10和11上的瓷砖。

瓷砖y轴坐标的计算稍微复杂一些：

```
pos.y = (int)((tileMap.mapSize.height * tileMap.tileSize.height - pos.y) / tileMap.tileSize.height);
```

请注意，我们使用括号的目的是让除法在最后一步执行。如图10-5所示，tileMap.mapSize被设置为30x20块瓷砖。tileMap.tileSize（每块瓷砖的大小）是32x32像素。我们的具体计算如下：

```
pos.y = (int)((20 * 32 – 260) / 32)
```

将tileMap.mapSize.height（地图纵向的瓷砖数量）与tileMap.tileSize.height（单块瓷砖的像素高度）相乘得到的是整个瓷砖地图的高度值（以像素表示）。用整个瓷砖地图的高度像素值减去当前触摸位置的y轴坐标值260，我们可以得到当前触摸在瓷砖地图上的y轴位置（以像素表示）。而因为得到的高度值是用像素表示的，所以要除以tileSize.height（单块瓷砖的像素高度）。将得到的浮点数转换成整数以后，最终得到触摸瓷砖的y轴坐标。

CCLOG和NSAssert两行代码可以帮助我们在调试窗口看到计算结果，同时确保不会得到错误的瓷砖坐标。

优化和可读性练习

因为瓷砖地图的尺寸不会变化，所以我们可以通过在类的头文件中添加一个用于储存瓷砖地图高度（以像素表示）的成员变量，来优化瓷砖坐标的计算：

```
float tileMapHeightInPixels;
```

然后你可以在init方法中的瓷砖地图加载完成之后，计算得到tileMapHeightInPixels变量的值：

```
CCTMXTiledMap* tileMap = [CCTMXTiledMap tiledMapWithTMXFile:@"orthogonal.tmx"];
tileMapHeightInPixels = tileMap.mapSize.height * tileMap.tileSize.height;
```

最后，你可以重写之前的代码，在每次调用tilePosFromLocation方法时，节省一次乘法操作：

```
pos.y = (int)((tileMapHeightInPixels - pos.y) / tileMap.tileSize.height);
```

上述优化不会为你赢得最佳优化的大奖，因为这只是个很细微的性能优化。不过每个优化都是有用的，而且使用易于理解的变量名会让我们的计算代码更容易读懂。

使用物体层（Object Layer）

我为本章制作的瓷砖地图，orthogonal.tmx瓷砖地图，包含了一个物体层，命名为ObjectLayer。你可以在Tiled软件中，通过选择Layer > Add Object Layer来添加物体层。然后点击瓷砖地图内的某个区域，画出长方形的物体层。不过我觉得“物体层”这个名称有些误导人，因为大多数游戏把这些长方形区域作为趣味点和触发区域来使用，而不是作为实际的物体来使用。

在Tilemap03项目中，我给ccTouchesBegan方法添加了一些新的代码，用于配合物体层的使用。列表10-4展示了相关部分的代码。这些代码是紧跟着isWater检测代码：

列表10-4. 检查触摸是否发生在ObjectLayer的长方形区域内

```
// 检查触摸是否发生在其中一块长方形区域内
CCTMXObjectGroup* objectLayer = [tileMap objectGroupNamed:@"ObjectLayer"];
```

```
bool isTouchInRectangle = NO;
int numObjects = [objectLayer.objects count];
for (int i = 0; i < numObjects; i++)
{
	NSDictionary* properties = [objectLayer.objects objectAtIndex:i];
	CGRect rect = [self getRectFromObjectProperties:properties tileMap:tileMap];
	if (CGRectContainsPoint(rect, touchLocation))
	{
		isTouchInRectangle = YES;
		break;
	}
}
```

因为物体层是一种不同的层，所以你不能利用瓷砖地图（tilemap）的layerNamed方法得到它。cocos2d中的物体层由CCTMXObjectGroup类表示，这个名字有点误导人，因为在Tiled软件中叫做object layer，而不是object group。你可以通过CCTMXObjectGroup类的objectGroupNamed方法得到物体层，需要提供的参数是在Tiled软件中物体层的命名，在我们的例子里是“ObjectLayer”。

接着，我用一个for循环遍历objectLayer.objects这个可变数组（NSMutableArray），此数组包含一列NSDictionary（字典）条目。听起来挺耳熟的？是的，这些属性字典和瓷砖地图的propertiesForGID方法返回的属性字典是一样的 — 唯一的区别是这些属性是在Tiled软件中定义好的，而且不能编辑。它们包含每一个长方形（层上的物体）的坐标信息。使用getRectFromObjectProperties方法可以得到指定的长方形物体：

```
-(CGRect) getRectFromObjectProperties:(NSDictionary*)dict
tileMap:(CCTMXTiledMap*)tileMap
{
	float x, y, width, height;
	x = [[dict valueForKey:@"x"] floatValue] + tileMap.position.x;
	y = [[dict valueForKey:@"y"] floatValue] + tileMap.position.y;
	width = [[dict valueForKey:@"width"] floatValue];
	height = [[dict valueForKey:@"height"] floatValue];
	return CGRectMake(x, y, width, height);
}
```

上述字典中的键值：x，y，width和height，都是在Tiled软件中设置的。我只是简单的把这些键对应的值通过valueForKey方法从字典中提取出来，然后用floatValue方法将NSString转换成浮点数。得到的x和y轴值需要加上tileMap位置的位移值，因为这些长方形物体会和瓷砖地图一起移动。最后，我们通过调用CGRectMake这个方法返回一个CGRect。

回到ccTouchesBegan方法。剩下的代码将会通过CGRectContainsPoint方法来检查触摸的位置是否落在长方形区域内。如果是，isTouchInRectangle这个布尔

标记变量就会被设置为YES，我们通过使用break来停止和跳出for循环，因为我们已经找到被触摸的长方形，不需要再检查其它区域了。然后，在ccTouchesBegan结束的地方，isTouchInRectangle变量会被用于决定是否在触摸的地方播放一个粒子效果。得到的效果就是：如果触摸是在其中一个长方形物体区域内，在手指触摸的地方就会播放一段粒子爆炸效果：

```
if (isTouchOnWater)
{
        [[SimpleAudioEngine sharedEngine] playEffect:@"alien-sfx.caf"];
}
else if (isTouchInRectangle)
{
        CCParticleSystem* system = [CCQuadParticleSystem particleWithFile: @"fx-explosion.plist"];
        system.autoRemoveOnFinish = YES;
        system.position = touchLocation;
        [self addChild:system z:1];
}
```

渲染物体层上的长方形区域

当你运行Tilemap03项目时，如图10-12所示，你会看到物体层上的长方形显示在瓷砖地图上。这不是瓷砖地图或者物体层的功能，而是我用OpenGL ES代码渲染出来的。每个CCNode都有一个叫做-(void)draw的方法，你可以重写这个方法，在其中添加自定义的OpenGL ES代码。我经常把用于碰撞测试和距离测试的线条，圆圈和长方形区域在地图上渲染出来，用于调试。将这些信息用可视化的方式显示出来，比在调试窗口中比较各个坐标要来得清晰的多。我们的大脑更擅长于处理可视化的信息，而不是数字。

图10-12. 通过OpenGL ES代码在瓷砖地图上显示物体层的长方形区域。

你只需要将-(void)draw方法放在瓷砖地图的类里面就可以了，它将在每一帧被自动调用。不过，你应该避免使用draw方法去修改节点的属性，因为这样会干

扰节点本身的渲染。**列表10-5**展示了TileMapLayer类中的draw方法。

列表10-5. 渲染 ObjectLayer 中的长方形

```
-(void) draw
{
	CCNode* node = [self getChildByTag:TileMapNode];
	NSAssert([node isKindOfClass:[CCTMXTiledMap class]], @"not a CCTMXTiledMap");
	CCTMXTiledMap* tileMap = (CCTMXTiledMap*)node;

	// 获取物体层
	CCTMXObjectGroup* objectLayer = [tileMap objectGroupNamed:@"ObjectLayer"];

	// 用OpenGL ES代码将线条宽度设置为3个像素
	glLineWidth(3.0f);
	glColor4f(1, 0, 1, 1);

	int numObjects = [[objectLayer objects] count];
	for (int i = 0; i < numObjects; i++)
	{
		NSDictionary* properties = [[objectLayer objects] objectAtIndex:i];
		CGRect rect = [self getRectFromObjectProperties:properties tileMap:tileMap];
		[self drawRect:rect];
	}

	// 将线条宽度设置为默认的1个像素
	glLineWidth(1.0f);
	glColor4f(1, 1, 1, 1);
}
```

首先，我通过tag获取到瓷砖地图（tileMap），然后使用tileMap的objectGroupNamed方法得到objectLayer（物体层）。接着，我用OpenGL ES的glLineWidth方法将线条宽度设置为3个像素，同时使用glColor4f将线条颜色设为紫色。对线条宽度和颜色的设置会影响之后代码中任何使用OpenGL ES方法渲染的线条 — 不仅仅在当前的方法中，其它节点中用于渲染的OpenGL ES代码都会受到影响（例如，任何在cocos2d的“CCDrawingPrimitives.h”头文件中定义的用于渲染线条，圆圈和多边形的方法）。所以在完成渲染以后，我重置了glLineWidth和glColor4f的设置。OpenGL是一个状态机，因此任何做了修改的设置都会被系统记住，从而影响之后渲染方法的输出结果。为了避免上述问题，你应该在使用完修改过的Open GL设置以后，将这些设置重新改回默认设置。

注：在-(void) draw方法中的代码进行渲染时使用的z-order值是0。而且此方法会在所有具有z-order为0的节点渲染之前进行渲染。这就意味着任何OpenGL ES代码渲染得到的图形将会被其它也拥有z-order为0的节点所覆盖。在我们渲染物体层里的长方形的代码中，我必须将tileMap的z-order值设为-1，这样才不会将一开始用OpenGL ES代码渲染的长方形覆盖掉。

和之前一样，我用for循环遍历所有物体层中的长方形，从字典中得到它们的属性以获取指定物体的CGRect，然后将CGRect传给drawRect方法。不幸的是，cocos2d移除了这个便利方法（drawRect）。不过我们可以自己使用ccDrawLine方法来实现自己的drawRect方法，如**列表10-6**所示：

列表10-6. 渲染一个长方形

```
-(void) drawRect:(CGRect)rect
{
        // 长方形是由四个线条组成的
        CGPoint pos1, pos2, pos3, pos4;
        pos1 = CGPointMake(rect.origin.x, rect.origin.y);
        pos2 = CGPointMake(rect.origin.x, rect.origin.y + rect.size.height);
        pos3 = CGPointMake(rect.origin.x + rect.size.width, rect.origin.y + rect.size.height);
        pos4 = CGPointMake(rect.origin.x + rect.size.width, rect.origin.y);

        ccDrawLine(pos1, pos2);
        ccDrawLine(pos2, pos3);
        ccDrawLine(pos3, pos4);
        ccDrawLine(pos4, pos1);
}
```

我为长方形的四个角都生成一个CGPoint，然后我将这四个点应用于ccDrawLine方法中以连接两个点生成一条线。你应该将此方法保存到可以方便找到的地方，因为你会经常使用此方法。

你会注意到这里的draw和drawRect方法都被 #ifdef DEBUG 和 #endif 声明所包含。这意味着这些物体层上的长方形不会在发布时被显示，因为这些长方形只会被用于调试和展示之用，所以最终用户不应该看到它们。

```
#ifdef DEBUG
-(void) drawRect:(CGRect)rect
{
        ...
}
-(void) draw
{
        ...
}
#endif
```

滚动瓷砖地图

我们在本章的最后部分讨论瓷砖地图的滚动。实际上我们可以很轻松的实现地图的滚动，因为我们只需要让CCTMXTiledMap移动就可以了。在Tilemap04项目的ccTouchesBegan方法中，当得到触摸到的瓷砖的坐标以后，我调用了centerTileMapOnTileCoord方法：

```
-(void) ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
        ...
        // 由触摸位置信息获取相应瓷砖的坐标
        CGPoint touchLocation = [self locationFromTouches:touches];
        CGPoint tilePos = [self tilePosFromLocation:touchLocation tileMap:tileMap];

        // 移动瓷砖地图，让触摸到的瓷砖处于屏幕中央
        [self centerTileMapOnTileCoord:tilePos tileMap:tileMap];
        ...
}
```

列表10-7展示了centerTileMapOnTileCoord方法，它会移动地图，让触摸到的瓷砖处于屏幕中央。当地图的任意边界与屏幕边缘对齐时，地图将会停止滚动：

列表10-7. 移动瓷砖地图，让触摸到的瓷砖处于屏幕中央

```
-(void) centerTileMapOnTileCoord:(CGPoint)tilePos tileMap:(CCTMXTiledMap*)tileMap
{
        // 获取屏幕大小和屏幕中心点
        CGSize screenSize = [[CCDirector sharedDirector] winSize];
        CGPoint screenCenter = CGPointMake(screenSize.width * 0.5f, screenSize.height * 0.5f);

        // 瓷砖的坐标0，0点是在左上角
        tilePos.y = (tileMap.mapSize.height - 1) - tilePos.y;

        // 此变量用于移动瓷砖地图
        CGPoint scrollPosition = CGPointMake(-(tilePos.x * tileMap.tileSize.width),
                                             -(tilePos.y * tileMap.tileSize.height));

        // 把得到的scrollPosition加上位移值
        scrollPosition.x += screenCenter.x - tileMap.tileSize.width * 0.5f;
        scrollPosition.y += screenCenter.y - tileMap.tileSize.height * 0.5f;

        // 确保地图边界和屏幕边界对齐时让地图的移动停止
        scrollPosition.x = MIN(scrollPosition.x, 0);
        scrollPosition.x = MAX(scrollPosition.x, -screenSize.width);
        scrollPosition.y = MIN(scrollPosition.y, 0);
        scrollPosition.y = MAX(scrollPosition.y, -screenSize.height);

        CCAction* move = [CCMoveTo actionWithDuration:0.2f position: scrollPosition];
        [tileMap stopAllActions];
        [tileMap runAction:move];
}
```

首先，我们获取屏幕的中心位置。因为瓷砖的坐标是从左上角开始计算的（如图10-11所示），而屏幕的0，0点则在屏幕左下角，所以我修改了tilePos变量的y轴坐标。使用(tileMap.mapSize.height - 1) 是因为瓷砖坐标是从0开始计

算的，也就是说，如果地图的高度是10块瓷砖，在瓷砖坐标里是用0到9表示的。

接着，我们生成一个scrollPostion的CGPoint，瓷砖地图将移动到这个点上。计算scrollPosition的第一步是将得到的瓷砖坐标与瓷砖大小（32像素）相乘，得到具体的以像素表示的位置信息。你可能不理解为什么要把得到的值设为负数。这是因为如果我想让地图从右上角向左下角移动的话，我必须通过让地图的坐标值减小来达到目的。

如果将上述代码得到的scrollPosition值用于移动地图的话，得到的效果就是触摸到的瓷砖左下角会与屏幕的左下角对齐。

要让触摸到的瓷砖移动到屏幕中央，我们必须修改scrollPosition的值。接下去的代码给scrollPosition的x和y轴坐标分别加上了屏幕一半大小的宽度和高度。因为考虑到瓷砖本身是有大小的，所以要减去瓷砖（tileSize）的一半大小。

通过使用Objective-C自带的MIN和MAX宏命令，我们可以确保将scrollPosition的值保持在瓷砖地图的边界之内，因此不会显示任何超出地图边界的部分。MIN和MAX两个命令会分别返回传给它们的两个参数中的最小值和最大值。它们比使用if,else这样的条件判断要更简洁和更具可读性。

最后，我们使用CCMoveTo动作移动瓷砖地图，让触摸到的瓷砖处于屏幕中央。得到的最终结果是：地图会滚动到你触摸的瓷砖上，让瓷砖处于屏幕中央。你也可以用相同的方法让地图滚动到需要关注的瓷砖上 - 比如主角精灵的位置。

结语

现在你应该理解了瓷砖地图的概念.你也学习了如何在Tiled Map Editor中制作可在cocos2d游戏中使用的，带有多个层和属性的瓷砖地图。

虽然在cocos2d中加载和显示瓷砖地图很简单，但是当我们要获取瓷砖和物体层，对它们进行修改和读写它们的属性时，就变得不那么简单了。本章也介绍了如何获取触摸到的瓷砖的坐标信息，也学习了如何使用得到的瓷砖坐标让地图滚动，并且将触摸到的瓷砖移动到屏幕中央。

我甚至介绍了如果用OpenGL ES代码在瓷砖地图上渲染显示物体层的长方形，以用于调试。