第十一章
45度角瓷砖地图（Isometric Tilemaps）

你可以在45度角瓷砖地图中使用2D图形来表现3D场景，这就是为什么45度角瓷砖地图非常流行的原因。这种地图允许你用相对简单的图形和工具创造出拥有空间感的，相对真实的游戏世界。而且渲染2D图形比3D图形对硬件的要求要低的多。**图11-1**展示了我们将在本章制作的样例游戏。你将会控制一个忍者角色在游戏世界中潜行，他会避开碰到的墙和山，也会躲藏在某些物体后面，比如树木和仙人掌。

图11-1. 45度角瓷砖地图

注：本章使用的瓷砖集来自David E. Gervais，通过“创作共用许可”（Creative Commons License）发布。你可以通过以下网址下载他的大多数作品：http://pousse.rapiere.free.fr/tome/index.htm

设计制作45度角瓷砖

45度角瓷砖地图的游戏使用的是“三向图投影”（Axonometric Projection），这种投影可以给你从一个角度观看地图的印象，由此创造出视觉深度。“三向图投影”是个技术用语，意思是将一个旋转过的3D物体投射到一个2D的平面上，得到的2D图形虽然变得歪曲了，但是我们的大脑还是认为此图形是3D的。

对于瓷砖地图来说，如果你看一下**图11-2**，你会看到我们是如何使用90度角图片生成45度角瓷砖的。首先，正方形图片会被旋转45度角，然后在y轴方向将它缩小成典型的45度角瓷砖的钻石形状。

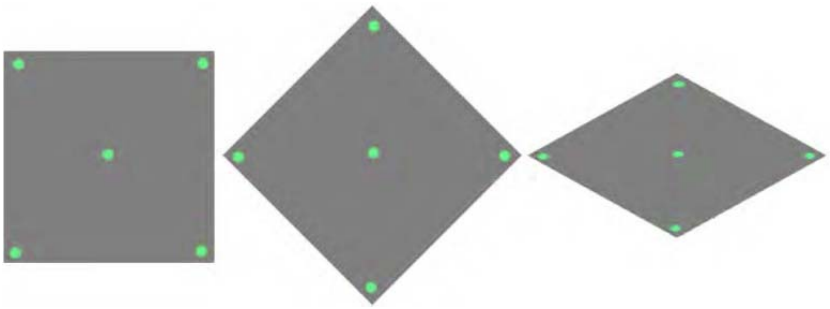

图11-2. 将90度角瓷砖旋转45度角，然后沿着y轴方向缩小成最终的45度角瓷砖

不过，**图11-2**只是在理论上可行。在实际操作中，你不能直接把90度角瓷砖直接旋转缩小成45度角瓷砖，因为旋转操作会影响图片里的内容。如**图11-3**所示，直接旋转缩小所得到的结果看上去还是2D的，而且视觉上完全不对。

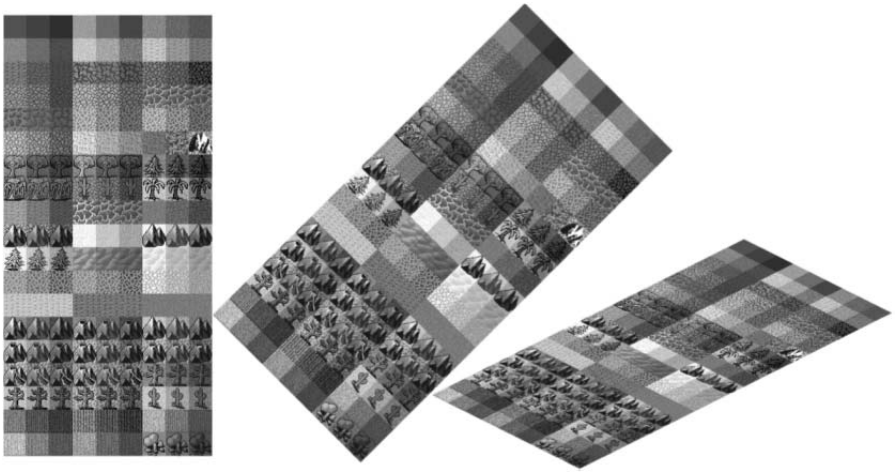

图11-3. 把90度角瓷砖转换成45度角瓷砖并不是那么容易的。

相反，你要把**图11-2**中的钻石状瓷砖作为你的画布。最容易设计的45度角瓷砖是平的地板瓷砖。你只要将上述钻石状瓷砖用一些图案填充就可以了。**图11-4**展示了由几块相连的45度角瓷砖排列而成的地板图案。虽然地板瓷砖并不那么引人注目，但是它们是游戏世界背景层中不可或缺的一部分。

图11-4. 45度角地板瓷砖看上去没有深度的感觉。它们在游戏中被用来填充背景地面区域。

要给45度角瓷砖地图添加视觉上的深度，你需要让瓷砖上画的物体超出钻石状瓷砖的边界。最常用的方法是将物体画成以45度视角观察的3D图形。而且要让物体超出瓷砖的边界，不过最多超出一块瓷砖的大小。你可以通过**图11-5**中的石门来理解上述方法。石门的上半部份超出了石门所在的瓷砖，延升到了上面一块瓷砖中，由此创造出视觉上的深度。

图11-5. 物体的高度超出本身所在的瓷砖，延升进入上面的瓷砖，以此创造视觉深度。

因为瓷砖是从后面开始往前渲染的，所以45度角瓷砖地图允许瓷砖相互叠加，这意味着离观看者近的物体瓷砖会叠加在离观看者远的瓷砖上，由此产生深度的感觉。不过这要求你仔细设计单独的瓷砖和瓷砖地图，因为太多的瓷砖叠加或者叠加了错误的瓷砖会很容易就破坏深度的感觉。

我建议你不要把形状类似的物体瓷砖叠加在一起，不过你应该使用相同的或者类似的配色方案。例如，在图11-5中，你不能把水晶瓷砖直接放在石门后面，因为这两者之间缺乏对比，而且它们的轮廓线条还会重叠，从而破坏深度的感觉。

同样，瓷砖上的物体高度不应该超出两块瓷砖的高度大小，如果超出就很难创造出令人置信的3D画面，因为玩家只能在同一时间看到瓷砖地图的一部分。比如，你画的大城堡的城墙有十几块瓷砖的高度，当主角走向城堡时，城堡的城墙很容易被认为是地板的一部分。45度角瓷砖不会由于离开屏幕远而变小，所以你可能会创造出如 M.C.Escher的作品所创造出的那种视觉幻象（你可以在这里看到M.C.Escher的作品：
http://www.mcescher.com/Shopmain/ShopEU/facsprints-nieuw/prints.html
）。

图11-6展示了名为dg_iso32.png的由David精心制作的45度角瓷砖集。它包含了多种地板瓷砖，墙壁，树木和房屋类的物体瓷砖，还有各种可被放在任何地板瓷砖上的装饰性物体。瓷砖集里的瓷砖大小是 54x49 像素。你可以选择不同的瓷砖高度；取决于你想要的瓷砖叠加程度，你可以选择大于或者小于49像素的瓷砖高度。瓷砖的钻石形状的实际高度是27像素。当你在 Tiled 软件中创造瓷砖地图时，知道钻石形状的高度很重要。

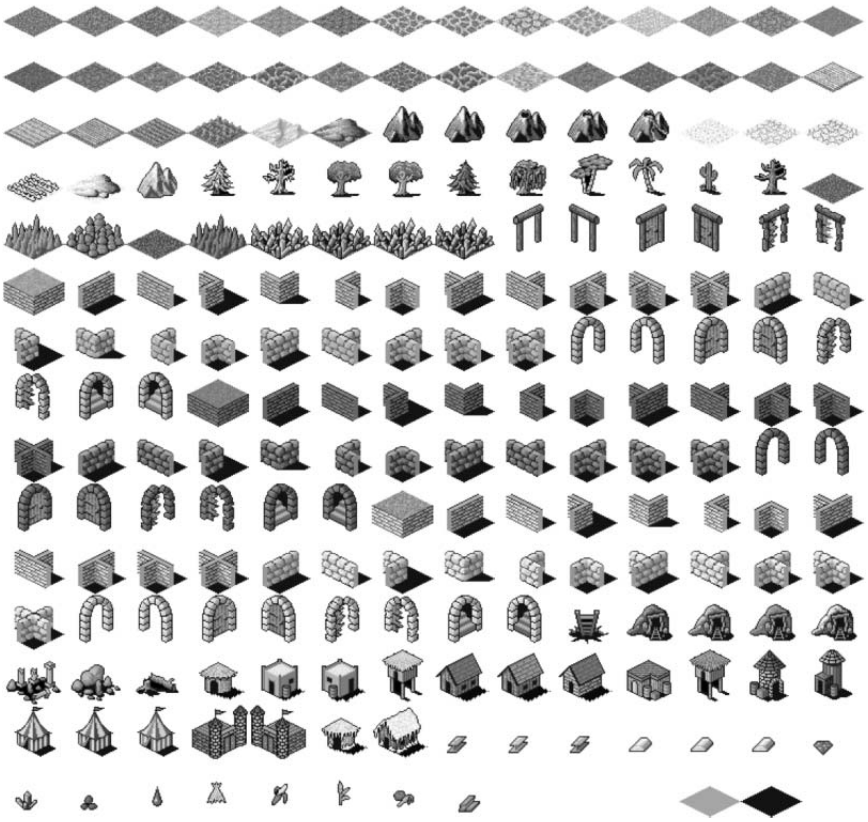

图11-6. David Gervais精心制作的45度角瓷砖集

使用Tiled编辑45度角瓷砖地图

我将使用Tiled Map Editor来制作45度角瓷砖地图。基本的地图编辑方法和90度角瓷砖地图一样，但是我们需要应用几个重要的步骤来正确设置45度角瓷砖地图和加载45度角瓷砖集。

生成一个新的45度角瓷砖地图

打开Tiled软件，选择File > New菜单会出现如**图11-7**所示的 New Map 对话框。显然，Orientation 应该被设置为 Isometric，Map size 的width（宽度）和height（高度）设为30块瓷砖。比较奇怪的一个设置是Tile size的宽度和高度设置。之前我提到过dg_iso32.png中的单块瓷砖的大小是 54x49 像素。而钻石形状的大小（当你将瓷砖放到地图中时，你要考虑这个尺寸）是 54x27 像素。但是我们在这里设置的却是 52x26 像素。

这里的像素差别是有意为之。因为45度角瓷砖设计的时候考虑到了它们会相互叠加，所以对于大多数45度角瓷砖集来说，当把它们应用到Tiled软件中的地图上时，地图上的瓷砖宽度必须比瓷砖集里的钻石形状的实际宽度小2个像素，高度要小1个像素。

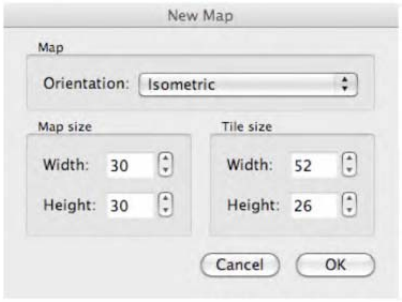

图11-7. 在Tiled中生成一个新的45度角瓷砖地图

使用上述瓷砖大小的像素值差别的目的是让瓷砖地图的边界成为一条直线，避免背景透过瓷砖而显示出来。这是必须的，因为不可能把钻石形状设计成可以让瓷砖之间的距离都相同，而又不相互叠加。

如果你看到如**图11-8**所示的缺陷，原因就是你在生成新45度角地图时设置了错误的瓷砖大小。你可以在Tilemap05项目的资源文件夹中找到这张出错的地图：isometricnooffset.tmx。

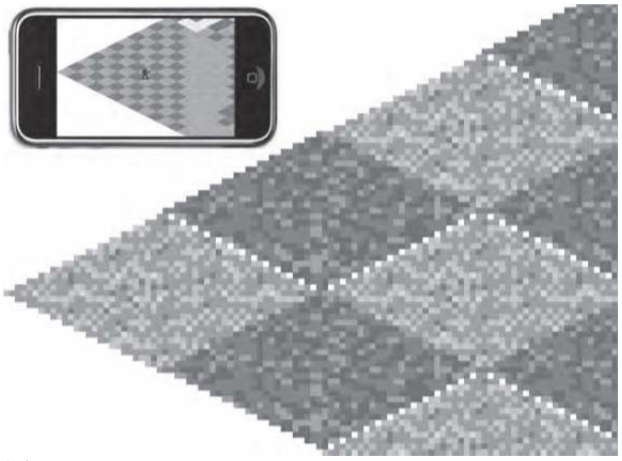

图11-8. 图中的地图缺陷是因为设置了错误的瓷砖尺寸。

如果你已经设置了错误的瓷砖尺寸，但是又不想丢失花费了好几个小时设计的地图，或者如果你出于其他的原因需要调整地图尺寸或瓷砖集尺寸，我教你一个简单的处理方法。以下技巧可以让你轻松试验不同的瓷砖尺寸，直到地图显示正确。在你的Xcode项目中选者 TMX 文件，你会看到它是个纯文本的XML文件。在文件的开头，找到地图的设置：

```
<map version="1.0" orientation="isometric" width="30" height="30" tilewidth="54" tileheight="27">
```

你可以在这里试验不同的 tilewidth 和 tileheight 参数，直到找到正确的瓷砖尺寸。同样的，如果你不能确定正在使用的45度角瓷砖集的瓷砖大小，你可以通过以下设置修改瓷砖集的 tilewidth 和 tileheight 参数：

```
<tileset firstgid="1" name="dg_iso32" tilewidth="54" tileheight="49">
        <image source="dg_iso32.png"/>
</tileset>
```

在你手动修改 TMX 文件以后，你要确保在 Tiled 里面重新加载 TMX 文件，因

为 Tiled 不会自动更新文件。

生成一个新的45度角瓷砖集

接着，需要在Tiled中加载一个包含45度角瓷砖的瓷砖集。我将在本章适用dg_iso32.png这个瓷砖集图片，你可以在Tilemap05项目的资源文件夹中找到这个图片。选择Tiled中的 Map > New Tileset… 菜单，然后找到 dg_iso32.png 文件。

如图11-7所示，你会注意到Tiled会在New Map对话框中为瓷砖设置默认的宽度和高度。对于45度角地图来说，由于瓷砖相互叠加的原因，默认设置是肯定要改变的。如前所述，dg_iso32.png瓷砖集所使用的瓷砖宽度是54像素，高度是49像素。请注意，在New Map对话框中使用的尺寸不是钻石形状的尺寸。图11-9展示了我们的瓷砖集的正确设置：

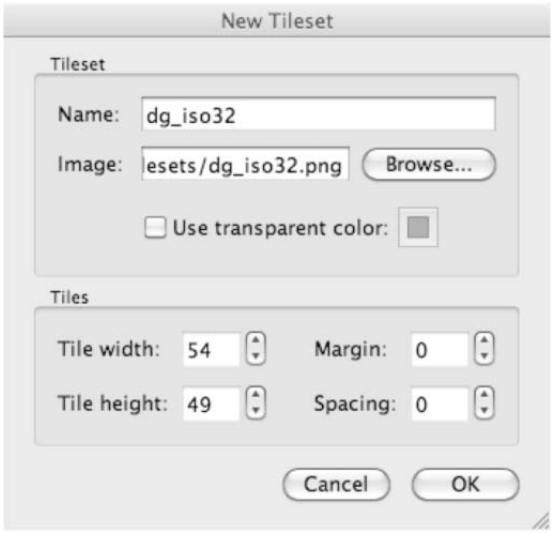

图11-9. 瓷砖集的宽度是54像素，高度是49像素

设计45度角地图的基本规则

设计45度角地图最重要的一条规则是：你需要为地图设计两个层，这样游戏角色才能在某些瓷砖后面走动。其中一层用来放地面上的物体和地板瓷砖，另一层用来放其它东西，例如与别的瓷砖重叠的瓷砖或者半透明的物品。在图11-6所示的瓷砖集里，头两行是地板瓷砖，可以放在Ground（地面）层。而第三行中的山和第4行中的大多数瓷砖都要放在Objects（物体）层中。

在Tiled中，通过选择 Layer > Add Tile Layer… 添加两个层，分别命名为Ground（地面）和Objects（物品）。Ground层应该放在Objects层之上。当你设计地图时，要注意在Ground层里只放置完全不透明的地板瓷砖。其它物体都要放在Objects层里。

在过去，cocos2d不能正确地在部份闭合的瓷砖后面显示游戏角色和其他精灵。一个解决方法是给Tiled中的层添加名为 cc_vertexz 的属性。我将在之后解释

这个解决方法；现在，选择Ground层，然后点击 Layer > Layer Properties…，在对话框中添加一个新的属性 cc_vertexz，将它的值设为 -1000。为Objects层添加相同的属性，不过将属性值设为 automatic，如图11-10所示。

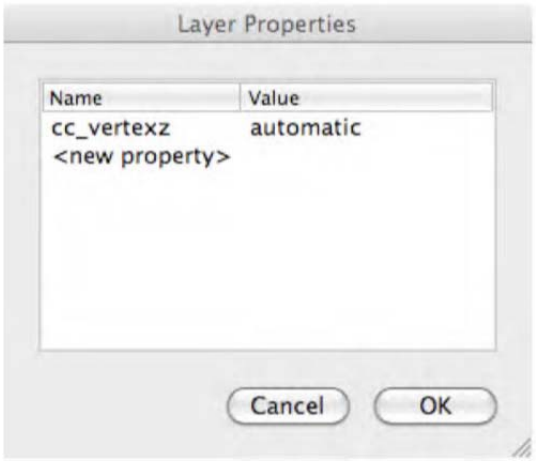

图11-10. Objects层的cc_vertexz属性要设为automatic

现在你可以花些时间设计一个好看的瓷砖地图了，或者也可以把Tilemap05项目中的地图加载进Tiled。请确保只在Ground层添加地板瓷砖，其它叠加和半透明或透明的瓷砖则放在Objects层。当你完成地图设计以后，你应该得到一个类似于如图11-11所示的地图：

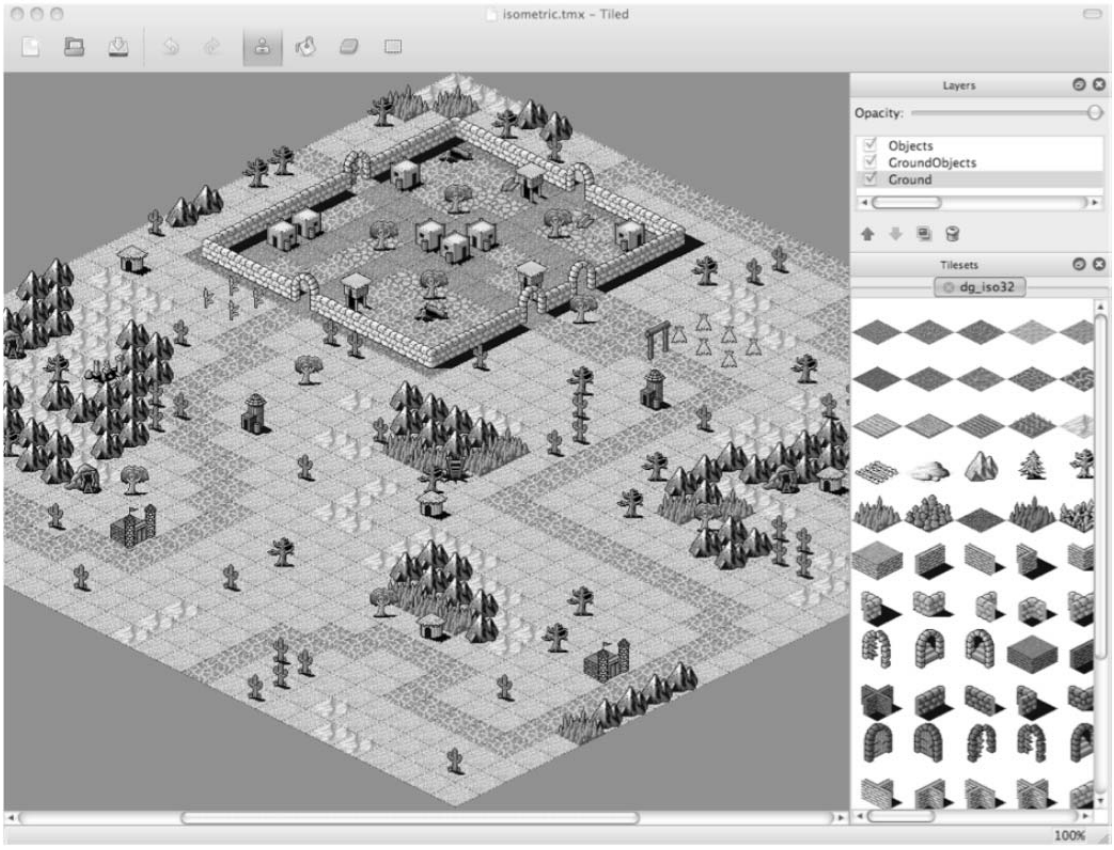

图11-11. 使用David Gervais的瓷砖集在Tiled中设计制作的45度角瓷砖地图

在cocos2d中使用45度角地图

让我们把45度角地图应用在cocos2d游戏中。你可能已经意识到，相比于使用90度角地图，我们要为45度角地图改变一些代码。特别是你要正确地设置cocos2d才能让游戏角色在部分被瓷砖挡住时，正确地显示在45度角瓷砖后面。用于寻找触摸到的瓷砖的代码与90度角地图也不一样。而且在滚动地图时，你也不能再在地图边界停止滚动了，因为45度角地图是钻石形状，而不是长方形的。

在cocos2d中加载45度角地图

加载地图很简单。与90度角地图相比，除了改变加载的文件名（isometric.tmx）外，你不需要改变任何代码：

```
CCTMXTiledMap* tileMap = [CCTMXTiledMap tiledMapWithTMXFile:@"isometric.tmx"];
[self addChild:tileMap z:-1 tag:TileMapNode];
tileMap.position = CGPointMake(-500, -300);
```

在上述代码中我直接将地图的位置设为（-500,-300）- 假设地图的尺寸是30x30 块瓷砖。如**图11-11**所示，这会让地图的上面部分的石墙的角落出现在屏幕中央。我之所以这样做是为了说明如何正确地在cocos2d中设置45度角地图，因为如**图11-12**所示，你可以看到加载的地图有明显的错误。

在cocos2d中正确设置45度角地图

如果你一直跟随我的步骤，而且在Tiled中给Ground和Objects两个层添加了cc_vertexz属性，你现在得到的地图应该类似**图11-12**中的地图。不知何故，Ground（地面）层缩小了好多，而Objects层的瓷砖则看起来像是漂浮在空中。整个场景看起来挺可怕的。

图11-12. 因为没有使用2D投影（2D Projection），渲染得到的Ground层出错了。

要解决上述问题，让cocos2d正确地渲染重叠的精灵，就需要让cocos2d使用不同于Xcode生成的应用模板所作的设置进行初始化。默认情况下，cocos2d模板会在程序代理的 applicationDidFinishLaunching方法中添加以下宏命令：

```
CC_DIRECTOR_INIT();
```

上述宏命令是在ccMacro.h中定义的，它会用标准的方式初始化cocos2d。虽然大多数游戏都可以用这种方式初始化cocos2d，但是在使用45度角地图时就行不通了。在Tilemap05项目中，我们用**列表11-1**中更为繁琐的代码来代替宏命令对

cocos2d进行初始化：

列表11-1. 手动初始化cocos2d的EAGLView

```
window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
if ([CCDirector setDirectorType:kCCDirectorTypeDisplayLink] == NO)
        [CCDirector setDirectorType:kCCDirectorTypeNSTimer];

CCDirector *director = [CCDirector sharedDirector];
[director setAnimationInterval:1.0/60];
EAGLView *glView = [EAGLView viewWithFrame:[window bounds]
                                pixelFormat:kEAGLColorFormatRGB565
                                depthFormat:GL_DEPTH_COMPONENT24_OES
                                preserveBackbuffer:NO];

[director setOpenGLView:glView];
[window addSubview:glView];
[window makeKeyAndVisible];

// 以下代码会修复地面层缩小的问题
[director setProjection:kCCDirectorProjection2D];
```

提供CC_DIRECTOR_INIT宏命令的原因是大多数游戏初始化cocos2d的时候所需的处理都是一样的。但是，当我们使用45度角地图的时候，我们需要在cocos2d初始化过程中修改一些EAGLView的参数，所以你就不能再使用宏命令了。需要改变的地方是：首先，你需要开启OpenGL的depth buffer（深度缓冲），这样就可以对物体的z ordering拥有更好的控制权。其次，CCDirector需要用到2D投影才能使用depth buffer。

首先，生成一个UIWindow。然后选择一个CCDirector类型，并且将动画时间间隔设置为60帧每秒。上述是默认的行为。

EAGLView那行代码很重要，因为要让重叠的瓷砖正确渲染的话，你必须用depthFormat指定一个depth buffer。在我们的例子里，我使用了GL_DEPTH_COMPONENT24_OES，这会生成一个24位的depth buffer。你可以通过将depth buffer设置为16位的来节省内存的使用量，因为16位可能也够用了。

深度缓冲可以让OpenGL有能力判断某个像素是在另一个像素之前或者之后，这样就可以判断是不是需要将新的像素渲染出来，或者丢弃。这会带来一些多余的内存开销 - 24位的depth buffer会占用500KB内存。但是这也允许精灵和瓷砖正确地重叠在一起。

glView生成以后被赋值给CCDirector。glView同时也被添加为 window 的一个子视图。最后整个window都被设置为可视。

另一行在初始化过程中很重要的代码是 setProjection，它会将cocos2d设置为

2D投影模式。这种投影模式会改变OpenGL的几个参数，从而影响cocos2d渲染节点的方式。在我们的例子中，这行代码解决了**图11-12**中的显示错误问题。如**图11-13**所示，现在所有图形都正确显示了。2D投影模式也允许你通过修改vertexZ属性对精灵的z-order进行细调，而不必通过精灵的zOrder属性来调节。我将在本章的后面部分再详细讨论。

图11-13. 使用了2D投影之后，地面瓷砖可以被正确显示了。

找到地图上触摸到的45度角瓷砖

接下去，我将会通过 Tilemap06 项目讨论如何找到触摸到的瓷砖在地图上的坐标。

如果你回顾一下上一章中的**图10-11**，你会记得90度角瓷砖地图的瓷砖坐标是从左上角开始算起的-也就是左上角是瓷砖地图的原点（0,0）。但是对于45度角地图，左上角这个原点不再存在了。因为地图被旋转了45度，所以地图顶部的那块瓷砖成了地图的原点。**图11-14**展示了45度角地图的坐标分布。地图往右下角的方向，瓷砖的X轴值在增加；往左下角方向的话，则瓷砖的Y轴值增加。在一张拥有30x30瓷砖的地图中，最底部那块瓷砖的坐标值就是（29,29）。

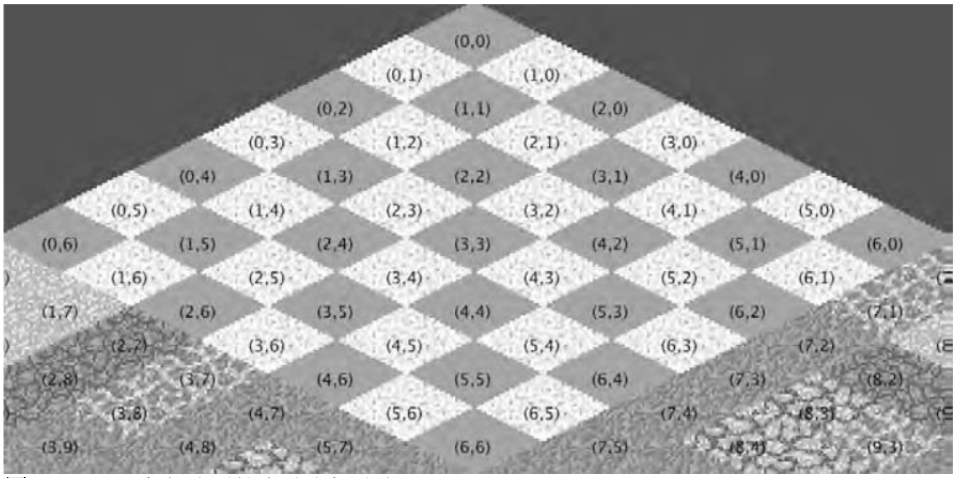

图11-14. 45度角地图的瓷砖坐标分布。

第一眼看到上述地图的坐标分布时你可能会觉得奇怪。但是如果你把头向右倾斜一点的话，你会看到其实现在的坐标分布是和90度角地图一样的，唯一的区

别是现在整张地图以顺时针方向旋转了45度角。

现在你可以把倾斜的头转回原来的位置了，因为我需要你专注于已被修改过tilePosFromLocation方法。此方法接受由触摸产生的屏幕位置信息，计算出触摸到的瓷砖坐标。如**列表11-2**所示，现在的代码比计算90度角地图的瓷砖坐标时所用的要复杂一些：

列表11-2. 由屏幕上的触摸位置信息计算瓷砖的坐标信息

```
-(CGPoint) tilePosFromLocation:(CGPoint)location tileMap:(CCTMXTiledMap*)tileMap
{
        // 触摸的位置信息必须减去瓷砖地图的位置信息，因为地图的位置可能在滚动变化
        CGPoint pos = ccpSub(location, tileMap.position);

        float halfMapWidth = tileMap.mapSize.width * 0.5f;
        float mapHeight = tileMap.mapSize.height;
        float tileWidth = tileMap.tileSize.width;
        float tileHeight = tileMap.tileSize.height;

        CGPoint tilePosDiv = CGPointMake(pos.x / tileWidth, pos.y / tileHeight);
        float inverseTileY = mapHeight - tilePosDiv.y;

        // 将得到的计算结果转换成 int，以确保得到的是整数
        float posX = (int)(inverseTileY + tilePosDiv.x - halfMapWidth);
        float posY = (int)(inverseTileY - tilePosDiv.x + halfMapWidth);

        // 确保坐标在地图的边界之内
        posX = MAX(0, posX);
        posX = MIN(tileMap.mapSize.width - 1, posX);
        posY = MAX(0, posY);
        posY = MIN(tileMap.mapSize.height - 1, posY);

        return CGPointMake(posX, posY);
}
```

考虑到地图的位置可能已经从一开始的原点位置移动过了，我们要把触摸的屏幕位置信息减去瓷砖地图的位置信息，这个步骤和90度角地图一样。接着我创建了6个变量，目的是让代码更具可读性，也可以减少打字量。tilePosDiv这个变量由瓷砖地图上的以像素表示的位置除以地图的宽和高计算得到。InverseTileY这个变量则是地图Y坐标的反转。这是因为瓷砖地图的Y坐标是从顶部开始的，而屏幕的Y坐标则是自下而上的。

现在我们来计算实际触摸到的瓷砖的X,Y轴坐标。因为45度角瓷砖地图刚好被旋转了45度角，所以我们可以使用比较简单的方法来计算出坐标点。我们首先计算反转过来的Y轴坐标值变量 inverseTileY，它的值将会在0到29之间（对于尺寸为30x30的瓷砖地图来说）.对于瓷砖的X轴值 posX，因为瓷砖的X轴向右延伸，

所以要将 inverseTileY 加上屏幕X轴坐标值除以瓷砖宽度的结果，再减去地图宽度的一半大小，也就是15（对于尺寸为30x30的瓷砖地图来说）。瓷砖Y轴值posY和posX的计算方法一样，所不同是因为瓷砖的Y轴向左延伸，所以我们inverseTileY要减去 tilePosDiv.x，然后加上地图宽度的一半大小。通过加上或者减去地图宽度的一半大小，我们在计算坐标时将地图的45度角旋转考虑了进来，因为现在的原点（0,0）是在地图顶部的中间位置，也就是地图宽度一半的位置。

注：我就不在这里详细解释上述计算所用到的数学概念了，因为你可以直接将上述代码应用到你的程序中。如果你有兴趣了解45度角投射的细节和它背后的数学原理的话，你建议你阅读Herbert Glaner所写的带有图解的文章：www.gandraxa.com/isometric_projection.aspx

通过使用 Objective-C 的 MIN 和 MAX 宏命令，我可以确保计算所得的坐标会在地图的边界范围之内。换句话说，如果我们使用的30x30的瓷砖地图，计算出来的坐标将会落在(0,0)和(29,29)之间。

移动45度角地图

完成对 tilePosFromLocation方法的更新以后，我们要在Tilemap06项目中使用tilePosFromLocation方法计算得到的瓷砖坐标，来实现45度角地图的移动。和90度角地图一样，我们在centerTileMapOnTileCoord方法中实现地图的移动。**列表11-3**展示了实现代码：

列表11-3. 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);

        // 获取地板层
        CCTMXLayer* layer = [tileMap layerNamed:@"Ground"];
        NSAssert(layer != nil, @"Ground layer not found!");

        // 仅在内部使用：瓷砖的Y坐标要减去1
        tilePos.y -= 1;

        // 获取瓷砖坐标处以像素表示的坐标信息
        CGPoint scrollPosition = [layer positionAt:tilePos];

        // 考虑到地图移动的情况，我将像素坐标信息乘以 -1，从而得到负值
        scrollPosition = ccpMult(scrollPosition, -1);

        // 为屏幕中央坐标添加位移值
        scrollPosition = ccpAdd(scrollPosition, screenCenter);
```

```
        // 移动瓷砖地图
        CCAction* move = [CCMoveTo actionWithDuration:0.2f position:scrollPosition];
        [tileMap stopAllActions];
        [tileMap runAction:move];
}
```

首先，我们需要获取屏幕大小和屏幕中心点。然后，我需要使用属于层的一个便利方法：positionAt。此方法会返回与瓷砖坐标相对应的屏幕位置信息。在使用positionAt方法之前，我需要先获取地板层（Ground Layer），并且通过NSAssert方法来确保得到的层确实是存在的。你可以使用地图中的任何一个层，前提是所有的层要使用相同尺寸的瓷砖。

在调用 positionAt方法之前，我必须将地图Y轴的值减去1，用以解决“persistent Offset”问题。有编程经验的读者可能会担心，如果Y坐标值为0的话，减去1以后，我们将会得到一个无效的负值索引。不过，positionAt方法并不是拿瓷砖坐标作为索引来使用的，这个方法可以使用任何类型的瓷砖坐标，包括负值的瓷砖坐标。

positionAt方法会返回与传入的瓷砖坐标相对应的像素坐标值，返回的坐标值被存储于 scrollPosition 变量中。这个方法并不只是设计给45度角地图使用的；它可被用于所有种类的瓷砖地图中：90度角地图，45度角地图和六边形地图。因为三种地图有很大的区别，所以cocos2d会首先检查当前使用的地图类型，然后再调用相应的代码进行计算。如果你有兴趣了解针对各种地图的计算方法，你可以在 CCTMXLayer.m 文件中查看以下三个方法： positionForOrthoAt，positionForIsoAt 和 positionForHexAt。

因为地图可能会移动，移动时地图的位置将会是负值，所以我将 scrollPosition 乘以-1，得到它的负值。然后我将得到的值与 screenCenter 位置相加，得到最终需要移动到的位置信息。这里的移动动作和之前的一样，它会移动地图，让触摸到的瓷砖处于屏幕中央。

处理45度角瓷砖地图的边界问题

因为45度角地图是菱形的，所以当地图移动时，显示出地图外的区域就无法避免了（如**图11-15**所示）。TilePosFromLocation方法可以确保返回的瓷砖坐标总是在地图边界之内，所以即使玩家触摸了地图外的区域，你也不用担心。不过如果你不想让玩家看到地图以外的区域，你必须使用一些技巧。

图11-15. 由于45度角地图是菱形的，所以移动到地图边界时，地图以外的一些区域也会显示出来。

打开 Tiled 软件，加载Tilemap06项目的资源文件夹中的isometric.tmx文件。你需要做的是给已有的地图添加一个边框，用瓷砖进行填充，从而让玩家觉得这是个不能穿越的区域。在Tiled中，使用 Map > Resize Map… 打开如**图11-16**所示的 Resize 对话框。在我们的例子中，我们需要为地图的四个边各添加10块瓷砖大小的区域以完全填满边界区域。取决于瓷砖的大小，你需要做些试验才能知道到底需要多少块瓷砖才能填满地图的边界区域。我在Width和Height输入框中输入了50，在Offset输入框中输入10。这些新设置会给地图增加20x20瓷砖大小的区域，同时软件会自动把原先设计好的地图移到新尺寸地图的中央，最终的结果是地图的每个边都会有10块地砖厚的边框。

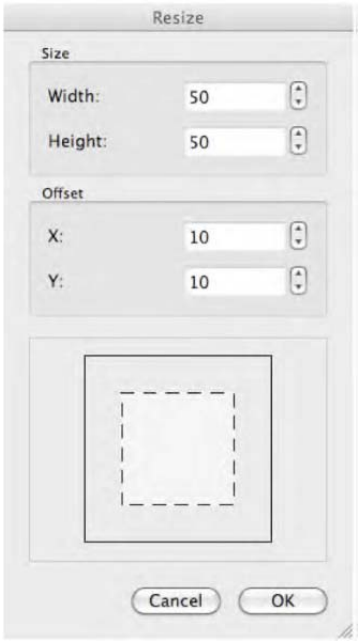

图11-16. 通过在Tiled中改变地图大小来增加一个边框区域。

现在，你可以在边框区域里填充各种瓷砖，让它看上去是不能穿越的。你可以选择深色的地板瓷砖，暗示玩家这里是不能穿越的区域。而且你还需要在边框

区域添加看上去无法穿越的物体，这些物体应该被放置在物体层。你得到的最终效果会类似于图11-17中的地图。我将制作完成的地图存在了Tilemap07项目的资源文件夹中，命名为 isometric-with-border.tmx。

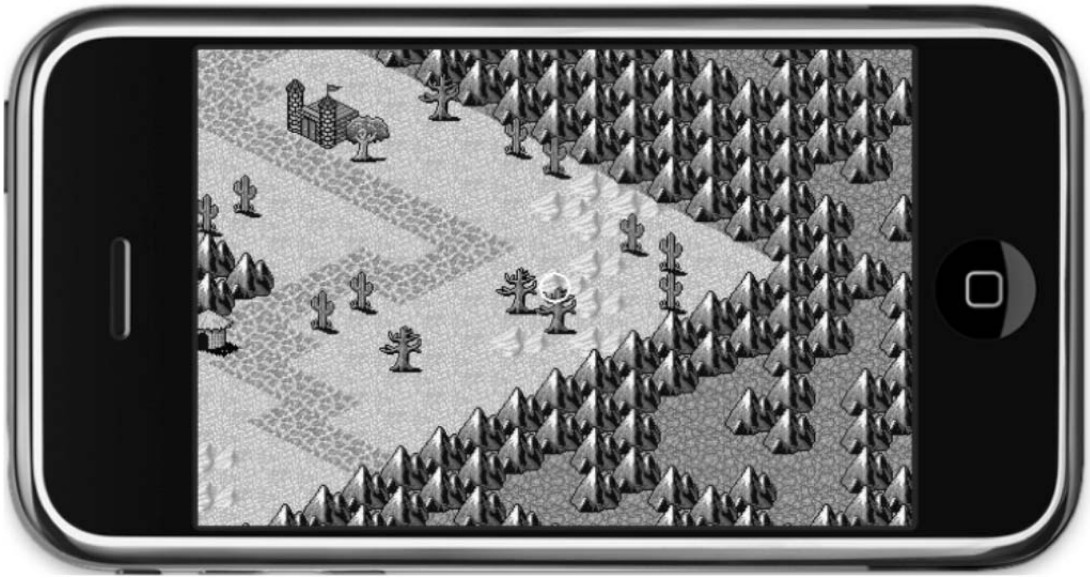

图11-17. 给瓷砖地图制作无法穿越的边框区域。

注：图11-17所示的无法穿越的区域看上有些重复和无聊。你可能想再多加些细节在上面。不过在上面添加细节是把双刃剑。从一方面来说，更多的细节和变化会让这块无法穿越的区域看上去更漂亮。从另一方面来说，有的玩家可能会认为区域内的某些地方是可以进去的，导致玩家花费无谓的时间了精力寻找进去的办法。如果有的玩家会这样想的话，这对你的游戏很不好。因为你不想让你的玩家把时间和精力浪费在寻找进入无法穿越的区域的方法上。

我在Tilemap07项目中添加了一些代码用于防止玩家移动到可玩区域（Playable Area）之外。我在 TileMapLayer 类中添加了两个CGPoint变量：playableAreaMin和playableAreaMax ：

```
@interface TileMapLayer : CCLayer
{
        CGPoint playableAreaMin, playableAreaMax;
}
```

在 TileMapLayer 类的初始化方法中，我用了10块地砖作为边框的尺寸来初始化上述两个变量：

```
const int borderSize = 10;
playableAreaMin = CGPointMake(borderSize, borderSize);
playableAreaMax = CGPointMake(tileMap.mapSize.width - 1 - borderSize,
                                                tileMap.mapSize.height - 1 - borderSize);
```

我将坐标(10,10)和(39,39)之间的区域定义为可玩区域（Playable Area）。所有位于这个区域之外的区域都被定义为无法穿越的区域。

剩下的事情就是更新 tilePosFromLocation 方法了。我们需要用playableAreaMin和playableAreaMax两个变量来替换 MIN/MAX 中的参数。目的是把玩家的活动范围局限在可玩区域之内，而不是之前的整个地图之内：

```
posX = MAX(playableAreaMin.x, posX);
posX = MIN(playableAreaMax.x, posX);
posY = MAX(playableAreaMin.y, posY);
posY = MIN(playableAreaMax.y, posY);
```

如果你现在运行游戏，你会看到只有可玩区域内的瓷砖在点击以后会被移动到屏幕中央。如果你点击了可玩区域之外的地方，你的点击也不会被忽略：地图会被移动到尽量靠近你点击的瓷砖区域。这样，玩家就会认为地图是无限延伸的，而不是只局限于可玩区域之内。

添加可移动的角色

通过添加一个可以在地图上移动的角色，我们离完成一个完整的45度角地图游戏就更近一步了。在我们的例子里，我选择了 ninjia.png这张图片来代表我们的角色，我将它添加到了Tilemap08项目中。我们的角色类继承自CCSprite，命名为Player。**列表11-4**展示了它的头文件：

列表11-4. Player类的头文件

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

@interface Player : CCSprite
{
}
+(id) player;
@end
```

列表11-5中的+(id)player方法是一个静态的自动释放的初始化方法。此方法中也利用了ninjia.png这张图片初始化了代表角色的精灵：

列表11-5. Player类的实现

```
#import "Player.h"
@implementation Player
+(id) player
{
        return [[[self alloc] initWithFile:@"ninja.png"] autorelease];
}
@end
```

接着，我们在 TileMapLayer 类的init方法中生成角色对象：

```
CGSize screenSize = [[CCDirector sharedDirector] winSize];

// 生成一个角色对象，把它添加到层中
```

```
player = [Player player];
player.position = CGPointMake(screenSize.width / 2, screenSize.height / 2);

// 调整角色贴图的位置，让角色处于所在地板瓷砖的中心点位置
player.anchorPoint = CGPointMake(0.3f, 0.1f);
[self addChild:player];
```

将角色的位置设在屏幕中央是有目的的。因为我们已经编写了让触摸到的瓷砖移动到屏幕中央的方法，通过将角色放置在屏幕中央，我们可以移动瓷砖地图，从而达到让角色看起来在移动的假象。也就是说，你不需要让角色移动，只要点击地图让地图移动就行了。

我把角色的定位点（anchorPoint）从默认的(0.5f,0.5f)修改成(0.3f,0.3f)，这样就可以把角色精灵的脚大致地放在所处瓷砖的中央位置。否则，角色的位置看上去就不大对了，因为所有其它游戏物体，比如树木和仙人掌的根部位置都是处于瓷砖中央的位置。

如果你现在运行游戏，虽然角色精灵保持在原地不动，但是当地图在移动时，看上去却是角色在地图上行走。

不过，如果你让角色移动经过山，墙壁，数目或者建筑物的时候，角色精灵总是出现在这些物体前面，而不是被它们所覆盖。

让角色移动经过物体瓷砖时出现在物体后面

要想让角色被在它前面的建筑物，墙壁，树木等物体挡住的话，你必须在角色移动时改变角色的vertexZ值。在本章开始的时候，当你在Tiled中创建物体层（Objects Layer）时，你给这个层设置了一个名为cc_vertexz的属性，并且将它的值设为automatic。上述设定会指示cocos2d为物体层的瓷砖分配连续的vertexZ值。图11-18展示了一张50x50尺寸的瓷砖地图上各个瓷砖分配到的vertexZ值。

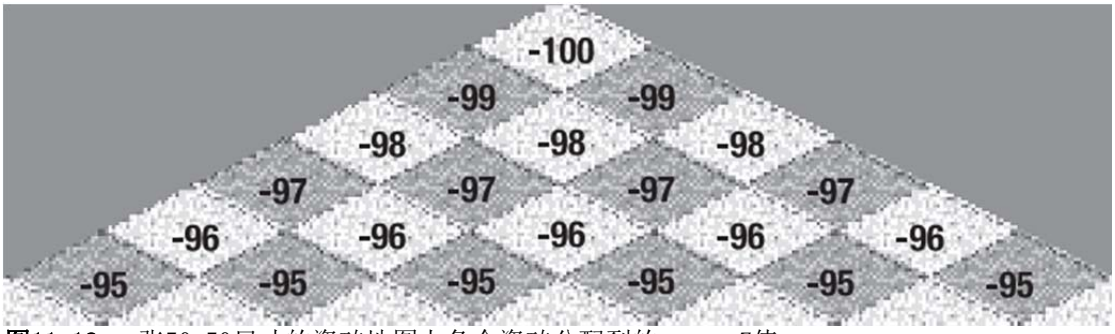

图11-18. 一张50x50尺寸的瓷砖地图上各个瓷砖分配到的vertexZ值

在Player类中，我添加了一个名为 updateVertexZ 的方法，用于分配各个瓷砖的vertexZ值：

```
-(void) updateVertexZ:(CGPoint)tilePos tileMap:(CCTMXTiledMap*)tileMap
```

```
{
        float lowestZ = -(tileMap.mapSize.width + tileMap.mapSize.height);
        float currentZ = tilePos.x + tilePos.y;
        self.vertexZ = lowestZ + currentZ - 1;
}
```

最小的vertexZ值是地图宽度和高度值之和的负值。同样的，你可以通过将指定瓷砖的X和Y轴值之和与最小的vertexZ值相加，得到与瓷砖相对应的vertexZ值。例如，坐标为(2,2)的瓷砖，X和Y轴之和为：2+2=4。将上述之和与最小vertexZ值相加：-100+4=-96。-96就是瓷砖的vertexZ值。因为在TileMapLayer类中，角色精灵是在加载了瓷砖地图以后再加载的，所以角色精灵会使用以所在瓷砖相同的vertexZ值进行渲染，从而覆盖在瓷砖上面。为了让精灵可以被物体瓷砖覆盖-也就是忍者可以躲到树木，建筑物等后面，我们要将角色精灵的vertexZ值减去1。因此，对于站在坐标为(2,2)，vertexZ值为-96的瓷砖上的角色精灵来说，他的vertexZ值就是-97（因为vertexZ值越小，相应的物体就离屏幕越远）。

要让上述 updateVertexZ 方法工作，你必须在Player类的头文件中定义此方法：

```
@interface Player : CCSprite
{
}
+(id) player;
-(void) updateVertexZ:(CGPoint)tilePos tileMap:(CCTMXTiledMap*)tileMap;
@end
```

然后，每次地图移动的时候，你都要调用 updateVertexZ 方法来更新角色精灵的vertexZ值。把调用此方法的代码放到TileMapLayer类的 ccTouchesBegan方法中可以达到这个目的：

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

        CGPoint touchLocation = [self locationFromTouches:touches];
        CGPoint tilePos = [self tilePosFromLocation:touchLocation tileMap:tileMap];

        [self centerTileMapOnTileCoord:tilePos tileMap:tileMap];

        //更新角色精灵的vertexZ值
        [player updateVertexZ:tilePos tileMap:tileMap];
}
```

如果你现在运行游戏，你会看到我们的忍者会躲藏在墙壁，树木和其它大点的物体后面了-就像真正的忍者一样。

在瓷砖之间移动角色（一块瓷砖一块瓷砖的移动）

到目前为止，我们触摸离开屏幕中心越远的区域，角色就移动的越快（其实是屏幕在移动）。而且角色可以自由的在瓷砖之间移动。但是正确的做法应该是让角色从一块瓷砖移动到另一块瓷砖，而且同一时间只能朝四个方向中的一个移动。我在Tilemap09项目中修改了用于控制角色移动的代码。现在，当你把手指放到屏幕上时，角色只会朝预先设定的四个方向中的一个方向移动。角色移动的方向取决于你在屏幕上触摸的位置与角色当前位置的相对关系。

如图11-6代码所示，这要求我们在TileMapLayer的头文件中添加一些新的代码：

图11-6. 添加了新代码的TileMapLayer类头文件

```
typedef enum
{
        MoveDirectionNone = 0,
        MoveDirectionUpperLeft,
        MoveDirectionLowerLeft,
        MoveDirectionUpperRight,
        MoveDirectionLowerRight,
        MAX_MoveDirections,
} EMoveDirection;

@interface TileMapLayer : CCLayer
{
        CGPoint playableAreaMin, playableAreaMax;
        Player* player;
        CGPoint screenCenter;
        CGRect upperLeft, lowerLeft, upperRight, lowerRight;
        CGPoint moveOffsets[MAX_MoveDirections];
        EMoveDirection currentMoveDirection;
}
```

我们将会使用EmoveDirection这个枚举类型来确定角色移动的方向，其中的MoveDirectionNone 表示没有任何移动。让我们来看一下列表11-7中对TileMapLayer类的init方法所做的修改：

列表11-7. 初始化角色（Player）的移动方向

```
// 将屏幕分成4块区域
screenCenter = CGPointMake(screenSize.width / 2, screenSize.height / 2);
upperLeft = CGRectMake(0, screenCenter.y, screenCenter.x, screenCenter.y);
lowerLeft = CGRectMake(0, 0, screenCenter.x, screenCenter.y);
upperRight = CGRectMake(screenCenter.x, screenCenter.y, screenCenter.x, screenCenter.y);

lowerRight = CGRectMake(screenCenter.x, 0, screenCenter.x, screenCenter.y);

moveOffsets[MoveDirectionNone] = CGPointZero;
moveOffsets[MoveDirectionUpperLeft] = CGPointMake(-1, 0);
moveOffsets[MoveDirectionLowerLeft] = CGPointMake(0, 1);
moveOffsets[MoveDirectionUpperRight] = CGPointMake(0, -1);
moveOffsets[MoveDirectionLowerRight] = CGPointMake(1, 0);
```

```
currentMoveDirection = MoveDirectionNone;

// 通过预约的更新方法来检查角色的移动
[self scheduleUpdate];
```

我定义了4个CGRect变量：upperLeft,lowerLeft,upperRight和lowerRight，用于将屏幕分成4个区域，点击其中一个区域时，角色就会向点击区域的方向移动。例如，如果玩家点击了屏幕的右下方，角色就会向地图的右下方移动。

moveOffsets这个数组包含与各个移动方向相对应的位移值。把此数组中的位移值与当前瓷砖坐标值相加后，我们就能得到相对应方向上的下一块瓷砖的坐标值。currentMoverDirection变量用于存储角色的当前移动方向。接着是预约方法 scheduleUpdate，用于连续不断地检查角色是否还要继续移动。

列表11-8中的ccTouchesBegan方法已经被修改过。其中的代码会被用于检查屏幕上4个区域中的哪一个接收到了触摸，然后设置currentMoveDirection变量的值到相应的方向。新添加的ccTouchesEnded方法将currentMoveDirection重置为MoveDirectionNone。

列表11-8. 让角色根据玩家在屏幕上的触摸位置进行移动

```
-(void) ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
        // 根据屏幕上的触摸位置信息，得到相对应的瓷砖坐标
        CGPoint touchLocation = [self locationFromTouches:touches];

        // 检查触摸出现在哪一个方向，从而设置角色的移动方向
        if (CGRectContainsPoint(upperLeft, touchLocation))
        {
                currentMoveDirection = MoveDirectionUpperLeft;

        }else if (CGRectContainsPoint(lowerLeft, touchLocation)){
                currentMoveDirection = MoveDirectionLowerLeft;

        }else if (CGRectContainsPoint(upperRight, touchLocation)){
                currentMoveDirection = MoveDirectionUpperRight;

        }else if (CGRectContainsPoint(lowerRight, touchLocation)){
                currentMoveDirection = MoveDirectionLowerRight;

        }
}

-(void) ccTouchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
        currentMoveDirection = MoveDirectionNone;
```

```
}
```

现在让我们专注与预约的更新方法上，游戏每一帧都会调用这个更新方法:

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

	// 如果地图正在移动，我们要等到它停止移动
	if ([tileMap numberOfRunningActions] == 0)
	{
		if (currentMoveDirection != MoveDirectionNone)
		{
			CGPoint tilePos = [self tilePosFromLocation:screenCenter tileMap:tileMap];
			CGPoint offset = moveOffsets[currentMoveDirection];
			tilePos = CGPointMake(tilePos.x + offset.x, tilePos.y + offset.y);
			tilePos = [self ensureTilePosIsWithinBounds:tilePos];

			[self centerTileMapOnTileCoord:tilePos tileMap:tileMap];
		}
	}

	// 持续不断地修改角色的vertexZ值
	CGPoint tilePos = [self floatingTilePosFromLocation:screenCenter tileMap:tileMap];
	[player updateVertexZ:tilePos tileMap:tileMap];
}
```

地图只有在移动时会有正在运行的动作（Action），所以我们可以通过检查地图正在运行的动作数量来判断地图是否在移动。只有地图不在移动和currentMoveDirection的值为MoveDirectionNone时，我们才为地图设置移动动作。因为角色总是处于屏幕中央，所以传给tilePosFromLocation的参数不再是屏幕上玩家的触摸位置，而是屏幕的中央位置 screenCenter。

角色所处的瓷砖位置tilePos加上moveOffsets数组中相应的移动方向的位移值（CGPoint类型），得到的计算结果是角色需要移动前往的下一个瓷砖的坐标。因为上述计算得到的坐标可能已经超出了可玩区域（playable area），所以我们将此坐标传给 ensureTilePosIsWithinBoungs方法，以确保得到的瓷砖坐标落在可玩区域之内（方法中所用的代码和之前用于确保瓷砖坐标落在可玩区域内的代码是一样的，只不过我将这些代码包装成了一个方法，以避免代码的重复）。最后，我们调用centerTileMapOnTileCoord方法来移动地图，使屏幕的中央与目标瓷砖坐标对齐；这里也会添加地图移动所需的动作（Action）。

现在角色可以在地图上一块瓷砖一块瓷砖地移动了，所以我们可以一直更新角色的vertexZ值。之前，角色的vertexZ值直接设置成了目标瓷砖坐标，这会导

致角色会被他所经过的所有物体瓷砖所阻挡。现在，当角色在地图上移动时，我们可以持续不断地更新角色的vertexZ值，从而让角色的z位置信息更加正确，消除了之前Tilemap08项目中出现的瓷砖重叠问题。

角色和物体碰撞时让角色停止移动

最后，我不想让角色在移动时穿过墙壁和山脉（虽然我们的角色是个忍者，不过他还没有那么大的本事）。为了解决这个问题，我们需要在Tiled软件中给地图添加一个新层。选择Layer > Add Tile Layer…添加一个新层，命名为Collisions（碰撞），然后将层的Opacity滑块移动到中间的位置。现在，在瓷砖集（tileset）中选择一块瓷砖，这块瓷砖的颜色要与地图上瓷砖的颜色有很大的区别，因为我们将在这个新建层里画出碰撞区域，所以这些碰撞区域即使在低透明度的情况下也要很容易识别。

我选择了紫色瓷砖中的一种。右键点击你选择的瓷砖，在快捷菜单中选择 Tile Properties…。注意：上述命名并不存在与Tiled的菜单中，你只能通过右键点击一块瓷砖后才能得到Tile Properties这个命令。在如图11-19显示的Tile Properties对话框中，添加一个名为 blocks_movement 的属性，将它的值设为1。实际上我将在代码中忽略这个值，最重要的是blocks_movement这个属性有一个值存在。

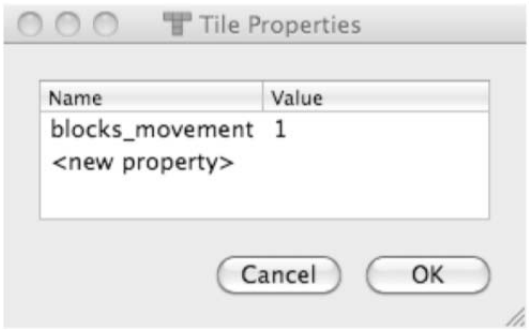

图11-19. 为瓷砖添加一个名为blocks_movement的瓷砖属性（Tile Property）

选中Collisions层，使用已经设置了blocks_movement属性的瓷砖，在地图上你不想让角色通过的地方画出瓷砖，比如墙壁，山脉，房子等等所处的位置。

Tilemap10项目中使用的地图isometric-with-border.tmx，已经存在一个碰撞层。碰撞层的唯一作用是用于测试是否某一块瓷砖应该允许角色通过，所以这个层不应该显示在游戏中。为此，我们所要做的第一件事情是在TileMapLayer类的init方法中将这个层设为隐形（代码如**列表11-9**所示）：

列表11-9. 隐藏碰撞层（Collisions Layer）

```
CCTMXTiledMap* tileMap = [CCTMXTiledMap tiledMapWithTMXFile:@"isometric-with-border.tmx"];
[self addChild:tileMap z:-1 tag:TileMapNode];

CCTMXLayer* layer = [tileMap layerNamed:@"Collisions"];
layer.visible = NO;
```

为了检查一块瓷砖是否被设置层了不可穿越，我在Tilemap10项目中添加了一个

isTilePosBlocked方法，如**列表11-10**所示：

列表11-10. 确认瓷砖是否已被设置为不可穿越

```
-(bool) isTilePosBlocked:(CGPoint)tilePos tileMap:(CCTMXTiledMap*)tileMap
{
	CCTMXLayer* layer = [tileMap layerNamed:@"Collisions"];
	NSAssert(layer != nil, @"Collisions layer not found!");

	bool isBlocked = NO;
	unsigned int tileGID = [layer tileGIDAt:tilePos];
	if (tileGID > 0)
	{
		NSDictionary* tileProperties = [tileMap propertiesForGID:tileGID];
		id blocks_movement = [tileProperties objectForKey:@"blocks_movement"];
		isBlocked = (blocks_movement != nil);
	}

	return isBlocked;
}
```

上述代码首先尝试着从碰撞层中获取指定瓷砖坐标处的瓷砖。如果不存在瓷砖，tileGID将被设为0，我们可以认为这块瓷砖没有被设置成不可穿越。但是如果我们在指定的瓷砖坐标找到了一个有效的tileGID，那么我们就会在地图上查询相应瓷砖的属性，此属性会以NSDictionary（字典）对象的形式返回。通过使用字典的 objectForKey 方法，如果名为 blocks_movement 的键（key）返回了一个有效的对象，那么我们可以确定这块瓷砖被设置成了不可穿越。

我们检查碰撞的地方是在预约的更新方法中，如**列表11-11**代码所示：

列表11-11. 在更新方法中检查碰撞

```
-(void) update:(ccTime)delta
{
	…
	// 如果地图正在移动，我们要等到它停止移动
	if ([tileMap numberOfRunningActions] == 0)
	{
		if (currentMoveDirection != MoveDirectionNone)
		{
			CGPoint tilePos = [self tilePosFromLocation:screenCenter tileMap:tileMap];
			CGPoint offset = moveOffsets[currentMoveDirection];
			tilePos = CGPointMake(tilePos.x + offset.x, tilePos.y + offset.y);
			tilePos = [self ensureTilePosIsWithinBounds:tilePos];

			if ([self isTilePosBlocked:tilePos tileMap:tileMap] == NO)
			{
```

```
                    [self centerTileMapOnTileCoord:tilePos tileMap:tileMap];
                }
            }
        }
        ...
}
```

在移动地图之前，我们调用 isTilePosBlocked 方法检查角色将要前往的瓷砖是否允许穿越。如果角色将要前往的瓷砖允许穿越，那么角色就会移动到目标瓷砖，否则角色就会停下来。

为游戏添加更多内容

到目前为止，我们可以在指引角色在45度角地图上移动。角色会躲藏在树木等物体后面，会避开碰撞只是这类游戏的基础。如果我们想在这个游戏世界中添加更多角色，比如敌人或者NPCs呢？

原则上来说，移动其它角色和移动主角角色的方式是一样的，唯一区别是我们的主角角色总是处于屏幕中央，而NPCs则会出现在地图上的任何一个地方。尽管如此，你只需要确认NPC将会移动的方向，然后像在centerTileMapOnTileCoord方法中移动层一样让NPC移动。唯一的区别是：因为你不是在移动层，而是移动NPC，所以运行的动作（Action）是针对NPC而不是层的，移动的方向也和移动层的时候相反。

一旦你让NPCs移动起来，接下去的步骤是如何让他们从A移动到B，同时避开障碍物和找到最短的路径。答案是：A*路径寻找算法（A* pathfinding algorithm）。这是业界使用的标准算法，已在很多情况下被采用和修改。这个路径寻找算法特别适用基于瓷砖的游戏，因为角色的位置通常受限于瓷砖的坐标。对A*路径寻找算法的详细介绍和许多其他游戏编程相关主题的介绍，你可以访问Amit的A*页面：http://theory.stanford.edu/~amitp/GameProgramming/.

你也可以访问Amit的游戏编程信息（Game Programming Information）页面。此页面链接到其他与人工智能和基于瓷砖的游戏相关的文章，包括程序自动生成游戏世界。很多文章可能看上去已经过时了，但是实际上大多是不受时间影响的，它们所提供的信息仍然很有用：
http://www-csstudents.stanford.edu/~amitp/gameprog.html.

结语

你在本章学习了45度角瓷砖地图的特殊之处，设计45度角地图的方法和创建一定深度的瓷砖地图。你也学习了如何在Tiled软件中添加一个不可穿越的边框区域和防止碰撞的方法。

我们也学习了如何设置45度角地图以配合cocos2d，如何设置cocos2d的2D投

影和深度缓冲 (Depth Buffer)以正确渲染重叠的瓷砖和精灵。

最后，我们在地图上添加了一个角色，设置角色的vertexZ属性，让角色正确地显示在其它瓷砖前面或者后面。我们学习了如何通过触摸屏幕，让角色一块瓷砖一块瓷砖地向手指触摸的方向移动。如果角色移动到我们在Tiled中设置的阻挡瓷砖前面，角色将会停止移动。

目前为止，我们的游戏中的元素都需要通过分开的步骤进行控制和动画。你需要为所有角色的移动，旋转，包括碰撞测试编写代码。在接下去的两章中，我将介绍物理引擎。物理引擎可以让游戏物体自己到处弹跳和相互碰撞。如果这是你第一次接触物理引擎，你将会经历魔术般的感受。抓紧你的帽子吧！