从零开始Android游戏编程（第二版） 第八章 地图的设计和实现

第八章 地图的设计和实现

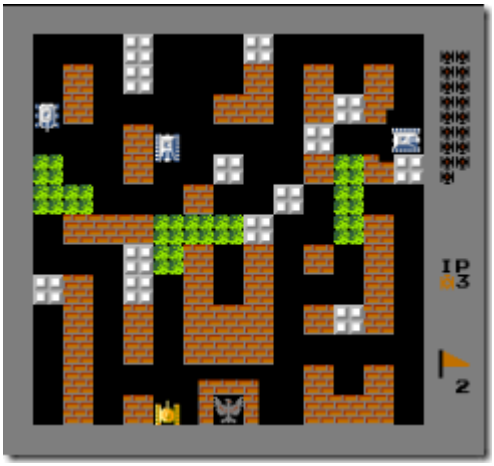
这本来是第十章，前面计划还 有两章的内容，一是跟第四章一样，完成一个 Asteroid 游戏作为小结，总结一下前面讲过的 Sprite 的用法，并演示 NPC 和子弹的处理方法。但是， 在写第七章的最后一个例子的时候，把本来简单的碰撞检测的例子扩充了一下，加入了 NPC 和子弹，基本和 Asteroid 的功能差不多了，所以就把原定的 Asteroid 砍掉了。另外一章是讲解程序的生命周期，内容相对简单，但考虑到上一章讲 Sprite 时已经引入了 TiledLayer，如果接着讲地图 应该会更连贯些，所以把生命周期向后移了一章。

那么，就先让我们看看地图的设计和实现。

如果我们需要的地图很小又很 少，完全可以将整个地图画在一张图片上。但是如果地图很多，绘制和管理地图的工作就会很麻烦，这时我们就需要用到另外一种技术——图块（Tile）。所谓 Tile，就是将地图中的公共元素提取出来，然后在显示的时候组合这些元素形成完整的地图，这就是本章要介绍的主要内容。

如下图是组成坦克大战地图的所有元素（16x16 像素）：

接下来让我们看一幅游戏中的场景：



可以看到，整个游戏场景就是由上面那些 Tile 构成的。这样，摆在我们面前的任务就很简单了：将地图依照 Tile 的大小分成若干格，将对应的 Tile 填到格子中。

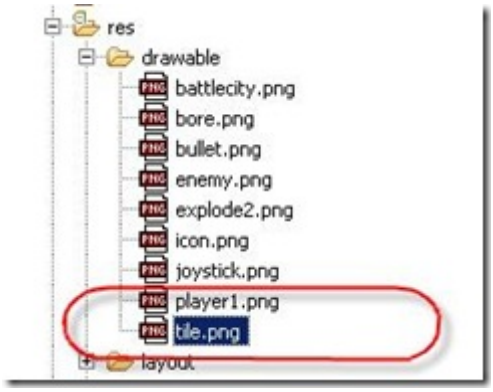
在 前面讲 Sprite 的时候，我们知道可以将组合在一张图片中的关键桢编号，以后就可以通过编号来使用这个桢（参看上一章桢动画的相关内容），这种方法也同 样适用于 Tile。而 2D 地图很容易让我们想到二维数组。也就是说我们可以将 Tile 的编号放到二维数组中，这样我们就可以通过历遍数组元素，像显示 Sprite 那样将整张地图一块一块的显示出来。

以上面的那张游戏截图为例，一共 13 行 13 列，我们可以定义一个 13x13 的二维数组（因为地图上有空白区域，所以我们将 Tile 的编号从 1 开始，用 0 表示空白）。

让我们找到 GameView_Old.java，增加成员变量 map[][]：

```
int[][] map = {

{0,0,0,2,0,0,0,2,0,0,0,0,0},

{0,1,0,2,0,0,0,1,0,1,0,1,0},

{0,1,0,0,0,0,1,1,0,1,2,1,0},

{0,0,0,1,0,0,0,0,0,2,0,0,0},

{3,0,0,1,0,0,2,0,0,1,3,1,2},

{3,3,0,0,0,1,0,0,2,0,3,0,0},

{0,1,1,1,3,3,3,2,0,0,3,1,0},

{0,0,0,2,3,1,0,1,0,1,0,1,0},

{2,1,0,2,0,1,0,1,0,0,0,1,0},

{0,1,0,1,0,1,1,1,0,1,2,1,0},

{0,1,0,1,0,1,1,1,0,0,0,0,0},

{0,1,0,0,0,1,1,1,0,1,0,1,0},

{0,1,0,1,0,1,6,1,0,1,1,1,0},

};
```

在资源中增加 tile.png



在构造函数中初始化 bitmap 对象

```
res = context.getResources();

bmp = BitmapFactory.decodeResource(res, R.drawable.tile);
```

在 onDraw 中绘图

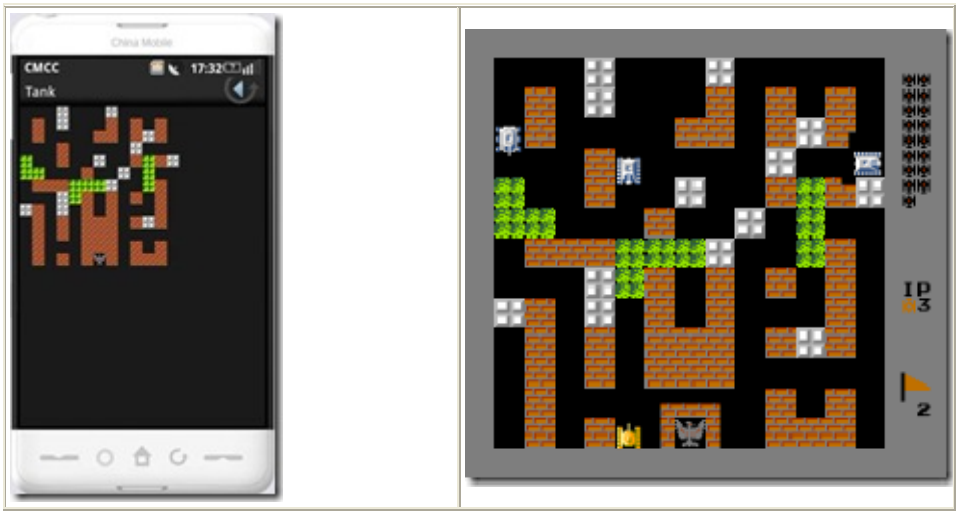
```
//用来显示图块的Rect对象

Rect src = new Rect(0, 0, 0, 16);

Rect dst = new Rect();

for(int i=0; i<13; i++) {

for(int j=0; j<13; j++) {

//根据 Tile 的编号得到对应的位置

src.left = (map[i][j]-1) * 16;

src.right = src.left + 16;

//根据地图上的编号计算对应的屏幕位置

dst.left = j * 16;

dst.right = dst.left + 16;

dst.top = i * 16;

dst.bottom = dst.top + 16;

canvas.drawBitmap(bmp, src, dst, paint);
```

```
}

}
```

最后不要忘了修改 Main.java 中的 setContentView

```
GameView_Old gameView;

/** Called when the activity is first created. */

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

gameView = new GameView_Old(this);

setContentView(gameView);

}
```

好了，运行一下程序看看结果：



对比一下原图，除了基地四周的砖块之外两者并无二致，可以说我们的 Tile 地图实践基本成功。

现 在我们知道了使用 Tile 显示地图的原理，实际上，我们不需要每次都很麻烦的写那么多代码，还记得前面说过的 TiledLayer 么？其中早已封装了上述 操作。不仅如此，TiledLayer 还能显示动态的地图呢。下面就让我们看一看 TiledLayer 的基本用法。其实，它与 Sprite 的用法非常相似（我们需要将 TiledLayer.java

加入到项目中，请使用本章附带程序的 TiledLayer.java 文件，上一章的 TiledLayer 并不 能正确工作）：

这次让我们使用 GameView，把刚刚的数组 map 拷贝到 GameView 中，并声明一个 TiledLayer 类型变量

```
//背景

TiledLayer backGround;
```

在构造函数中初始化 TiledLayer

```
// 背景图

backGround = new TiledLayer(13, 13, BitmapFactory.decodeResource(res,

R.drawable.tile), 16, 16);
```

TiledLayer 的构造函数有 5 个参数,分别是地图的行列数(是以 Tile 为单位的),包含 Tile 的 bitmap 对象，Tile 的宽度和高度。

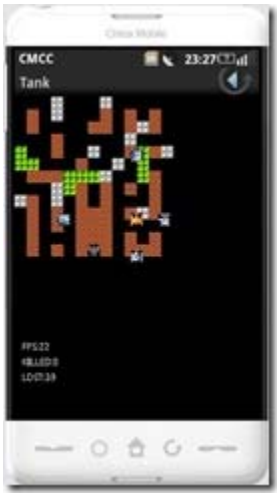
通过 setCell 方法将定义在数组中的 Tile 编号传递给 TiledLayer。

```
for(int i=0; i<13; i++) {

for(int j=0; j<13; j++) {

backGround.setCell(i, j, map[i][j]);

}

}
```

最后,只需要在 run 函数中调用 paint 方法,就可以将 TiledLayer 显示出来了。当然，你可以像控制 Sprite 那样控制 TiledLayer 显示的位置。

```
backGround.paint(c);
```

让我们看一下运行的效果



下面让我们来学习如何实现动态地图。所谓动态地图跟前面讲到的桢动画是一个道理，就是循环显示几个关键桢。让我们看一下前面 Tiles 的图片

我们会发现有两张水域的图片，这就是为动态地图准备的，组合起来之后应该会有如下的效果：

那么，我们如何在 TiledLayer 中实现动态地图呢？TiledLayer 为我们准备了这样几个函数：

createAnimatedTile： 创建动态图块。很多人会迷惑于这个函数的名字，说是创建动态图块，可是创建在哪儿啊？创建出来怎么用呢？只有天知道。其实，这个函数的主要功能也就是为动 态图块分配了一个存储结构。你不调用它还会报错，调用了其实也没什么用。createAnimatedTile 返回一个动态图块的编号，从-1 开始依次递 减，第一次调用返回-1，这样就分配了一个编号为-1 的动态图块。第二次调用会返回-2，依次类推。这个返回值一般没有用，因为我们做地图的时候肯定已经 确定了动态图块的位置，这个编号早就写到了数组中了。以后你就可以通过这个编号来控制相应的图块。函数有一个参数，指定一个 Tile 的编号，动态图块最初 显示的就是这个 Tile。而正是通过改变这个 Tile 来实现动画的。请看下面代码：

```
backGround.createAnimatedTile(4);
```

我们初始化了一个动态图块，编号是-1，参数 4 表示当前显示 Tile 序列图中的第四个 Tile，就是第一张水域的图片。

setAnimatedTile：动态图块的内容就是使用这个函数改变的。函数的第一个参数是动态图块的编号，如刚刚的-1。第二个参数是 Tile 的编号。

说到这里，大家应该清楚动态地图的用法了吧：

首先在地图数组中确定需要显示动态图块的位置，填入相应的编号，例如我们将上一张地图的第三行增加三块水域

```
int[][] map = {

{0,0,0,2,0,0,0,2,0,0,0,0,0},

{0,1,0,2,0,0,0,1,0,1,0,1,0},

{0,1,-1,-1,-1,0,1,1,0,1,2,1,0},

{0,0,0,1,0,0,0,0,0,2,0,0,0},

{3,0,0,1,0,0,2,0,0,1,3,1,2},

{3,3,0,0,0,1,0,0,2,0,3,0,0},

{0,1,1,1,3,3,3,2,0,0,3,1,0},

{0,0,0,2,3,1,0,1,0,1,0,1,0},

{2,1,0,2,0,1,0,1,0,0,0,1,0},

{0,1,0,1,0,1,1,1,0,1,2,1,0},

{0,1,0,1,0,1,1,1,0,0,0,0,0},

{0,1,0,0,0,1,1,1,0,1,0,1,0},

{0,1,0,1,0,1,6,1,0,1,1,1,0},

};
```

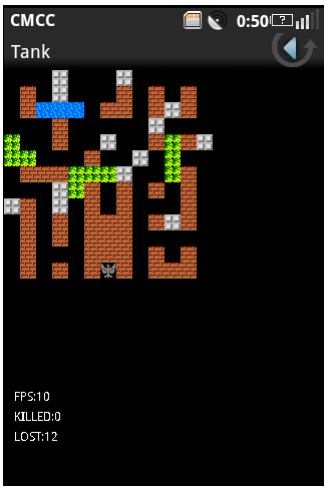
然后在 GameView 的构造函数中初始化 TiledLayer，除了在 setCell 之前调用 createAnimatedTile 之外，没有其他区别。

最后就是在 run 函数中调用 setAnimatedTile，不断地改变图块了

```
if(backGround.getAnimatedTile(-1) == 4) {

backGround.setAnimatedTile(-1, 5);

} else {
```

```
backGround.setAnimatedTile(-1, 4);

}

backGround.paint(c);
```

来让我们看一下运行的效果



其实笔者觉得 TiledLayer 完全也可以像 Sprite 那样使用桢序列和 nextFrame 来实现动态效果，似乎更易用一些，有兴趣的读者可以自己修改 TiledLayer 实现这个功能。

到这里，我们已经掌握了显示地图的方法，但是，这个地图还不能真正运用到我们的游戏中。读者肯定也看到了，在 TiledLayer 的第一个例子中，我们的坦克可以穿墙而过，显然，这个地图还缺少最基本的功能——阻挡。

有一种简单的方案可以实现阻挡，让我们看一下 Sprite 类，其中有一个方法：

```
public final boolean collidesWith(TiledLayer t, boolean pixelLevel)
```

检测 Sprite 与 TiledLayer 的碰撞。这种检测是以 Tile 为单位的，当与 Sprite 重合的 Tiles 编号不为 0 时函数返回 true，否则返回 false。

下面让我们做一个小例子测试一下：

打开 GameView_Old，增加一个 Sprite 类型的成员变量

```
// 主角

Sprite player;
```

在构造函数中初始化 player

```
// 初始化主角
player = new Sprite(BitmapFactory.decodeResource(res,
R.drawable.player1), 16, 16);
player.setFrameSequence(new int[] { 0, 1 });
```

在 onDraw 中绘制 player

```
player.paint(c);
backGround.paint(c);
```

在 onKeyDown 中控制 Sprite 的运动

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
// TODO Auto-generated method stub
switch (keyCode) {
case KeyEvent.KEYCODE_DPAD_UP:
x = player.getX();
y = player.getY();
player.move(0, -16);
if(!player.collidesWith(backGround, false)) {
y -= 16;
}
player.setTransform(Sprite.TRANS_NONE);
player.setPosition(x, y);
break;
case KeyEvent.KEYCODE_DPAD_DOWN:
x = player.getX();
```

```
y = player.getY();
player.move(0, 16);
if(!player.collidesWith(backGround, false)) {
y += 16;
}
player.setTransform(Sprite.TRANS_ROT180);
player.setPosition(x, y);
break;
case KeyEvent.KEYCODE_DPAD_LEFT:
x = player.getX();
y = player.getY();
player.move(-16, 0);
if(!player.collidesWith(backGround, false)) {
x -= 16;
}
player.setTransform(Sprite.TRANS_ROT270);
player.setPosition(x, y);
break;
case KeyEvent.KEYCODE_DPAD_RIGHT:
x = player.getX();
y = player.getY();
player.move(16, 0);
if(!player.collidesWith(backGround, false)) {
x += 16;
```

```
}

player.setTransform(Sprite.TRANS_ROT90);

player.setPosition(x, y);

break;

}

postInvalidate(); // 通知系统刷新屏幕

return super.onKeyDown(keyCode, event);

}
```

可以看到，坦克只能在空白区域运动，这回不能上墙了。

但 是这种方法还是比较粗糙的，很多时候不能实现我们的目的，比如坦克大战中，水和砖头是坦克不能通过的，但是掩体是可以通过的，还有子弹可以通过水域，这时 候还是检测 Tile 的编号来的准确些。就是说，我们事先确定好那些编号的 Tile 可以通过，哪些不能。然后模仿 collidesWith 方法根据 Tank 的位置取得它下一步要到达的 Tile 的编号，并判断坦克是否被阻挡。

让我们用这个方案重写 onKeyDown 方法（为了简化教程，我们假设每次 player 和 tile 都是完全重合的）：

首先我们先定义一个函数用来判断 Tank 是否可以通过

```
// 判断坦克是否可以通过

private boolean tankPass(int x, int y) {

// 不超过地图范围

if (x < 0 || x > 12 * 16 || y < 0 || y > 12 * 16) {

return false;

}

int tid = map[y / 16][x / 16];

if (tid == 1 || tid == 2 || tid == -1)

return false;
```

```
return true;
}
```

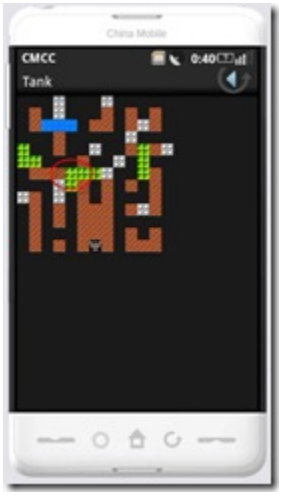
然后在 onKeyDown 中运用新的方案

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
// TODO Auto-generated method stub
switch (keyCode) {
case KeyEvent.KEYCODE_DPAD_UP:
x = player.getX();
y = player.getY();
player.move(0, -16);
if (tankPass(player.getX(), player.getY())) {
y -= 16;
}
player.setTransform(Sprite.TRANS_NONE);
player.setPosition(x, y);
break;
case KeyEvent.KEYCODE_DPAD_DOWN:
x = player.getX();
y = player.getY();
player.move(0, 16);
if (tankPass(player.getX(), player.getY())) {
y += 16;
}
```

```
player.setTransform(Sprite.TRANS_ROT180);
player.setPosition(x, y);
break;
case KeyEvent.KEYCODE_DPAD_LEFT:
x = player.getX();
y = player.getY();
player.move(-16, 0);
if (tankPass(player.getX(), player.getY())) {
x -= 16;
}
player.setTransform(Sprite.TRANS_ROT270);
player.setPosition(x, y);
break;
case KeyEvent.KEYCODE_DPAD_RIGHT:
x = player.getX();
y = player.getY();
player.move(16, 0);
if (tankPass(player.getX(), player.getY())) {
x += 16;
}
player.setTransform(Sprite.TRANS_ROT90);
player.setPosition(x, y);
break;
}
```

```
postInvalidate(); // 通知系统刷新屏幕

return super.onKeyDown(keyCode, event);

}
```

现在让我们运行一下看看吧，这回终于能够达到了我们想要的效果，我们的坦克正藏在掩体下面，并且它不能通过墙和水域。



到 此为止，关于地图的内容就讲解完毕了。这些内容并不复杂，首先介绍了使用 Tile 组成地图的原理，然后介绍了 TiledLayer 已经动态地图，最后演示 了阻挡的实现方法。本章的大部分例子使用了 GameView_Old，并没有使用游戏循环，也没有涉及到地图的平滑滚动，在以后需要的时候会补充这部分知 识。