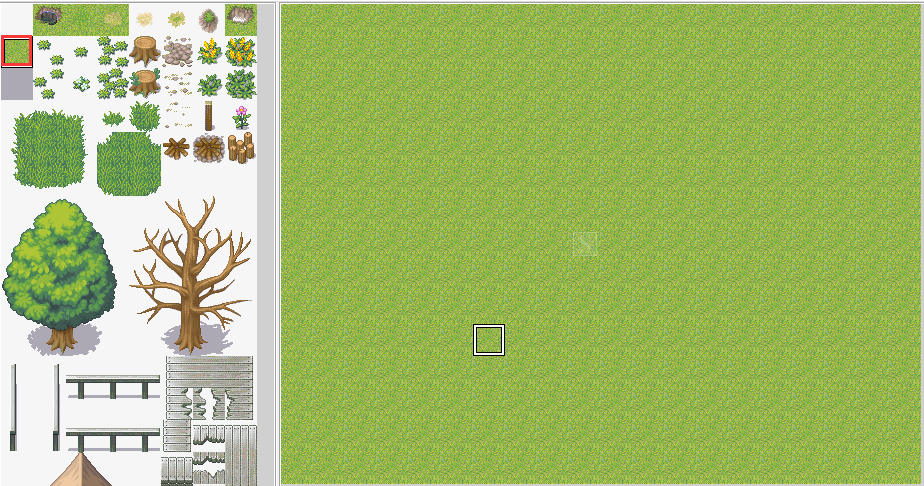
# 1.地图数据测试

## （1）测试地图与数据的对应关系

首先是新建工程后的默认地图 20 \* 15



它对应的是一个[20, 15, 3]的三维数组，每个数组元素的值为

[i, j, 0] = 384,

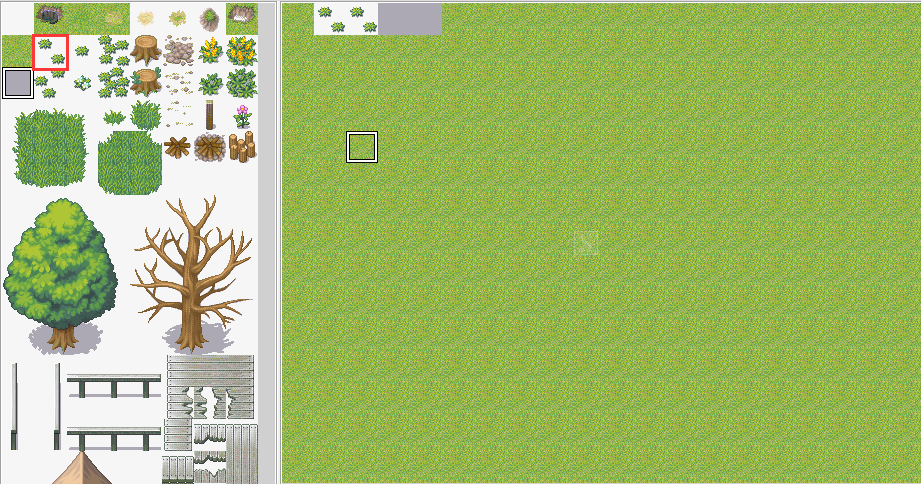
[i, j, 1] = 0,

[i, j, 2] = 0

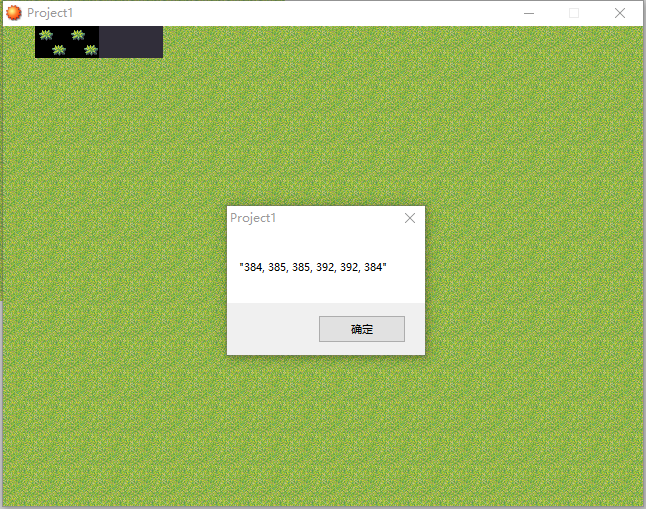
结论：由于默认地图中，只有第一层包含土块，而数组中的第二，三维（维度从左到右算起）的元素都是0，以及上数组各维长度与地图设置符合，所以基本可以肯定，第一维代表地图宽度，第二维代表地图高度。

## （2）测试各图块的值

使用第2行第2列的土块和第3行第一列的土块绘制，

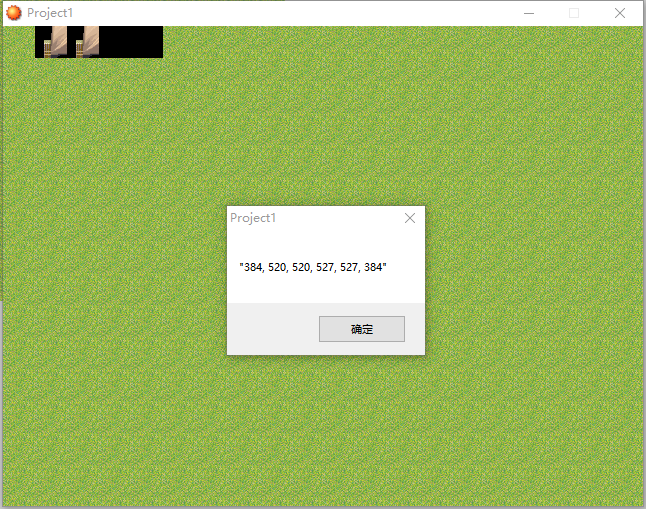


输入第一行的前6个土块对应的值，



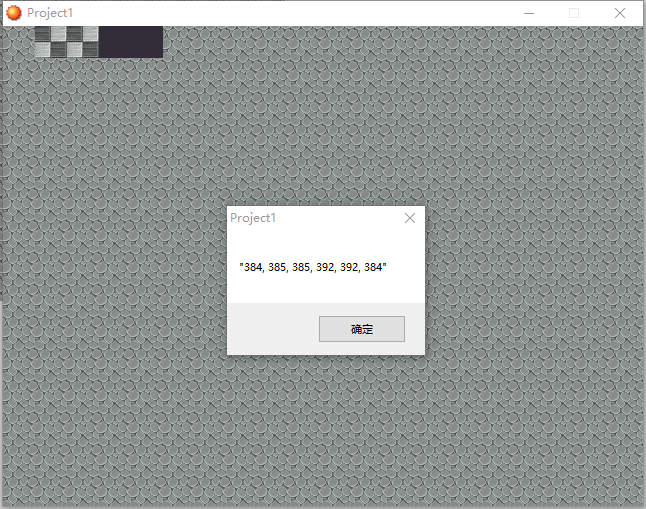
第2行第1列的图块对应384，第2行第2列的图块对应385，第3行第1列的土块对应392，而每行包含8个图块，384 + 8 = 392，符合显示的值。则，可以猜测，从第2行第1列开始，按从左到右，从上到下的顺序，图块对应的值依次增1.

检验：取最后一行（第19行）的第1个和最后1个绘制，则，它们的值应该分别是：(19 – 2) \* 8 + 384 = 520, 520 + 7 = 527.



符合预期。

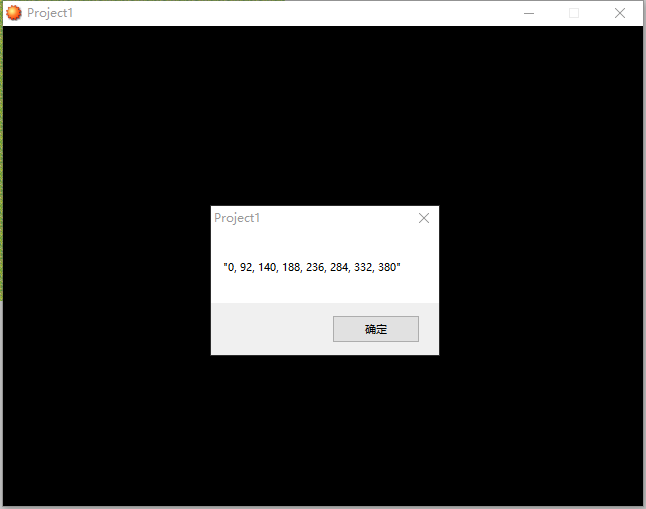
## （3）测试不同类型的地图元件的值是否和上面的一样

切换到文件名为033 天上世界 内部 的地图元件，同样绘制第二行第1，2列的图块和第3行第1列的图块，观察结果：

结论：地图元件的类型不会影响地图数组的值。

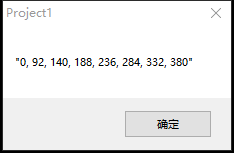
## （4）自动元件数值测试

绘制第一行的8个自动图块，并输出数值



从结果来看，从第2个值开始，每次都是递增48，直到380.第一个值是0也可以理解，因为第一个图块实际上没有对应任何自动元件文件，就是表示用来表示空白的。

接下来看看，在第2层绘制的结果，输出第2维的前八个数值[i, 0, 1], i = 1到8，

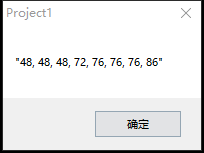


结果仍然没有变，看来只要绘制上去后，值就是固定的了。

## （5）自动元件组合测试

用同一个图块绘制一个2\*2的区域，观察该区域的值：





结果出乎意料，我还以为会是相同的呢。但是这也可以理解，从绘制出来的图就可以看出了，不同效果的方格的值是不同的，相同效果的方格的值是相同的。

那么，就需要找出具体的不同的算法。

已知，7个自动地图元件的编号分别开始于48，每个有48种值，具体参看标题6.

## （6）地图的绘制原理

红色代表重要概念

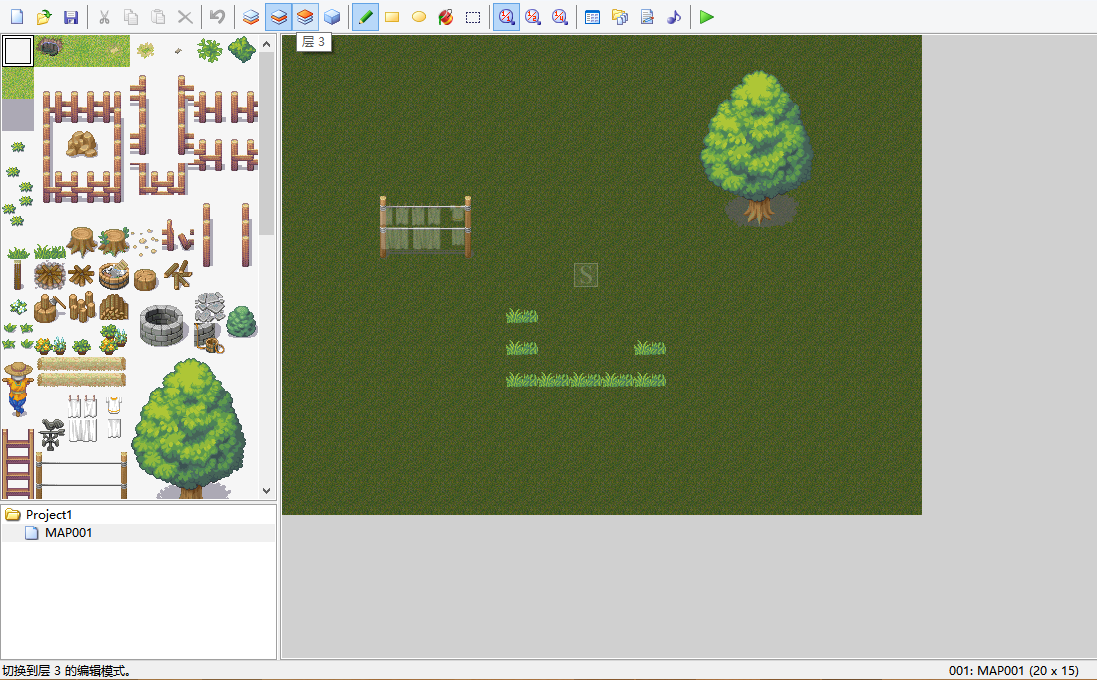
绿色代表提示

注意：文章中使用的代码均为JavaScript标准的伪代码。

要制作一个2D RPG游戏，所需的地图大的种类无非两种：TileMap、PicutreMap。

这一章，我们主要针对于RpgMakerXp所实现的TileMap做分析。

首先我们来看看RpgMakerXp的地图编辑界面。

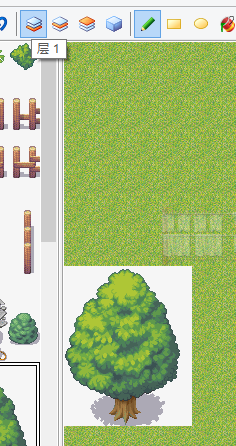


图块大小为32\*32，地图所使用图块数量为20\*15个格子。

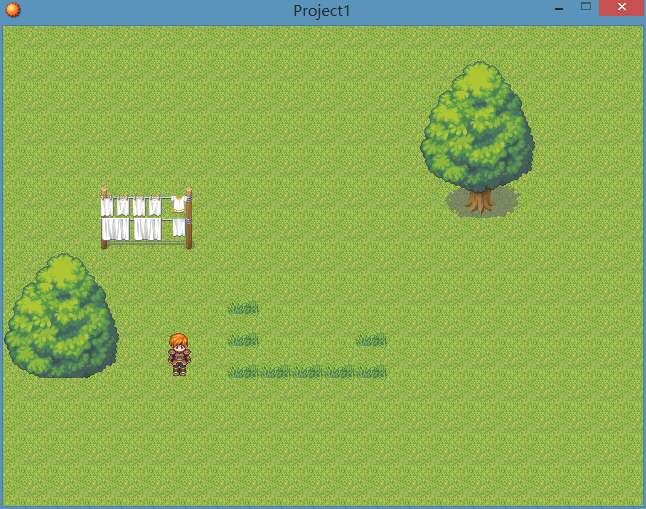
所显示的图层有3个。

一般来说我们编辑时候显示的内容就是最终显示的内容，但是运行后却发现，并不尽然。

如下情况：

我们先在图层1绘制一个树木然后我们在图层2再使用其他图块遮盖住这个树木

运行一下看看效果：

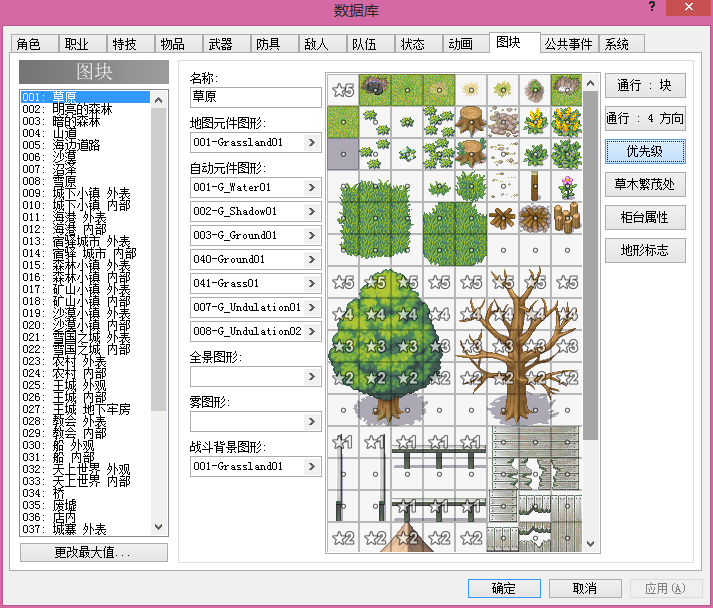


这是怎么一回事呢？我们不是用其他东西覆盖了吗？为什么还会出现？而且更为奇怪的是，为什么只显示了一半？

这里，我们就有了另一个概念，图块级别。

我们怎么知道图块级别是如何设置的呢？如果您的电脑里有RpgMakerXp的话，首先我们打开一个项目，然后选择菜单中的工具-数据库即可打开内部数据编辑界面。

我们使用的地图为001：草原，如下图所示，我们目前选择的是通行：块(即，图块的全局可通行设置)，手动选择到优先级(即，图块级别)。



这里我们能看到，树木的根部是没有优先级的，然后从下往上分别是L2、L3、L4和L5。下面我们就来看看，这些级别分别是用来干什么的。

假设，我们在图层1中的数据如下(以下数字代表绘制到地图上的图块级别，0为没有优先级):

5 5 5 5

4 4 4 4

3 3 3 3

2 2 2 2

0 0 0 0

图层2中的数据如下:

0 0 0 0

1 1 1 1

2 2 2 2

3 3 3 3

4 4 4 4

显示结果是如何的呢？按照RpgMakerXP的处理来说，流程会变成这样(相对于每个地图上要显示的格子来说，并非相对于使用的地图元件)：

1.对这个格子的3个图层根据图块级别排序，避免两个图层中图块级别一致导致绘制顺序错误(以下是该实现的伪代码)

1. data.layouts.sort(function (a, b) {
2. if (a.图块级别 > b.图块级别 || (a.图块级别 == b.图块级别 && a.图层 > b.图层)) {
3. return 1;
4. } else if (a.图块级别 < b.图块级别 || (a.图块级别 == b.图块级别) && a.图层 < b.图层) {
5. return -1;
6. } else {
7. return 0;
8. }
9. });

2.将排序好的数据依次绘制到对应的图块级别画布中(图块级别有5级，因此程序内建的画布应该也有5个)

3.将画布转换成图片进行缓存，以此来提高应用程序性能。

以上流程只会在地图加载的时候执行一次，以后均取得图片缓存直接绘制。

可是，为什么要这么复杂呢？

因为这样才能够实现人物与地图的遮挡关系，地图图块与图块之间的遮挡关系。

人物应该在最底层地图缓存之后绘制。

如此我们来测试一下。



完美！~

但是，还有一个问题我们没有解决。

那就是如何实现可通行判断？

其实这个非常简单，并不像地图绘制那么难。

但是这里还是要说一下的。

由于图层1、图层2与图层3中绘制的内容不一样。

那么就有可能在图层2中有一个不可通行块而图层1和图层3中没有的情况。

这种情况我们只需要这样简单的判断一下即可。

1. for(var i = 0; i < data.layouts.length; i++){
2. if(data.layouts[i].是否可通行 == false) return false;
3. }
4. return true;

# 2.Viewport视口测试

（1）创建一个标题图片的精灵，设置其视口为(100, 100, 100, 100)，结果是显示了原图像的(0, 0, 100, 100)区域，精灵画面的左上角移到了窗口的(100, 100)处。

（2）将精灵的x坐标改为100，y坐标改为100，精灵画面的左上角移到了窗口的(200, 200)处。

所以，在WPF中实现精灵时，精灵的实际坐标等于视口坐标加上x，y坐标

# 3.视口的测试

（1）测试视口的rect属性是否可以修改

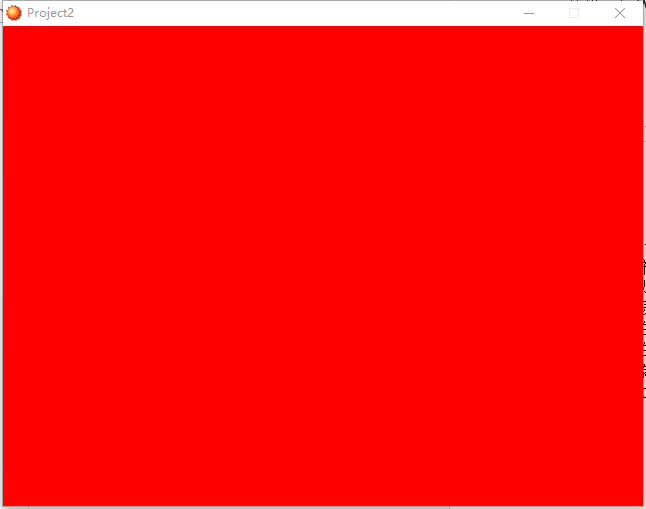
结果：可以修改

（2）将精灵视口的visible改为false，结果精灵不显示

（3）视口的z值优先级大于精灵的z值优先级

（4）视口的ox属性，设置为100，则精灵图像左移了100像素，设置为-100，则精灵图像右移了100像素

（5）将（0，0，640，480）的视口的color设置红色，则整个窗口变为红色：



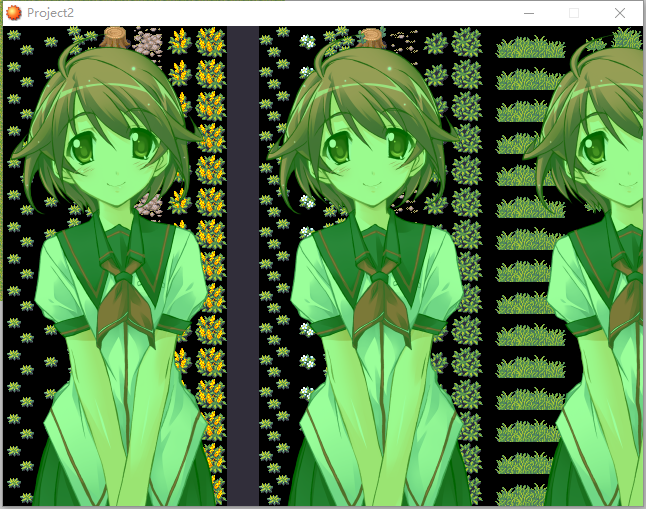
（6）将精灵的Tone设为(255, 0, 0)，则画面变为如下：



# 4.Plane测试

（1）Plane实际就是平铺精灵的效果

（2）Plane的color的属性，相当于在图像的上面覆盖了一层颜色板，使不透明的地方被颜色板挡住，透明的则无效果。以下是(0, 255, 0, 100)的color的效果。



# 5.RPG Maker XP地图路径

（1）可恶！RPG Maker XP安装目录下的Data文件夹下的Map001.rxdata不是工程制作的地图，而是默认地图。

（2）应该去工程目录寻找制作的地图对应的文件。

# 6.自动地图元件的绘制

参考网上的文章：

**序：**最近刚好想写个类似RMXP的地图编辑器，遇到的第一个问题就是自动地图元件的绘制问题。老实说，我不知道这东西到底叫什么（特别是英文叫什么），只知道RMXP翻译过来就是自动地图元件，而且魔兽的地图编辑器也用到这种技术，但我在网上却始终找不到相关的资料。其绘制原理其实很清晰，就是用户给出指定规格的地图元件资源，编辑器将其拆分，在绘制时再根据绘图上下文来进行拼接。但具体的过程以及拆分和拼接原理却很少有人提及，这里我整理了一下。

**1.RMXP的自动地图元件规格**

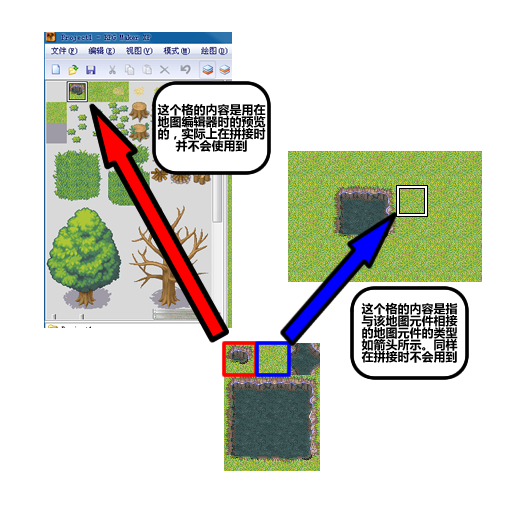
首先介绍下RMXP的自动地图元件规格，其中第一格和第二格都是展示用的，并不用在具体的拼接上。话说我也是刚搞懂的。

图1

真正用在拼接的就是剩下的内容，这里先将其按下图拆分开

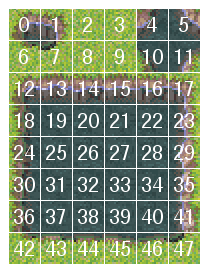


图2

这里的每一个小格就是最后用于拼接的格子，我们暂且叫它小元件，如图每一个小元件都有一个编号

虽然编号为0,1,2,3,6,7,8,9的小元件都不会用到，但这样可以方便由编号来进行定位小元件的矩形

计算编号为i的小元件在图中的矩形的方法为：

1. size = 16 // 小元件的宽高
2. row = i / 6 // 小元件所在的行
3. col = i % 6 // 小元件所在的列
4. (col \* size, row \* size, size, size)

**2.分析所有拼接情况**

在RMXP中，双击自动地图元件，即可以打开自动展开元件的对话框。里面显示的就是所有拼接情况。



图3

同样的，我们给每一种情况定一个编号

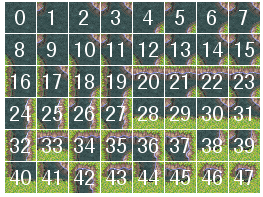


图4

注意，最后2个情况是一样的，所以总共有47种情况（编号从0开始）

每一种情况都是由4个小元件拼成的。这里先不急着确定每一种对应哪4个小元件。

先来分析下到底是哪47种情况。

假设我们现在正在用RMXP的地图编辑器来绘制自动地图元件，如下图，我们打算在位置4绘制一个自动地图元件

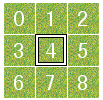


图5

那么怎么判断绘制出来的是哪个情况呢？很简单，根据周围的1,2,3,5,6,7,8格子是否有自动地图元件来确定位置4的绘制情况。（**这里要注意，影响位置4绘制情况的是周围8个格子是否有自动地图元件，而每个格子具体是哪个情况并不影响**）

按照排列组合公式，可以算出8个格子的出现组合有256种，这比47大多了。为什么呢？

这是因为RMXP并不支持对角连接，如下图

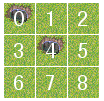


图6

1和4位置的自动地图元件并不连接起来，也就是说此时1位置并不会影响4位置的绘制情况。那什么时候才会影响呢？

看下图

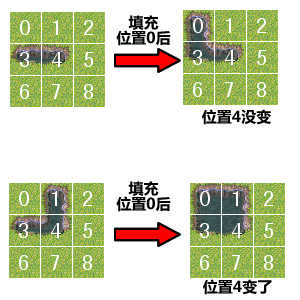


图7

可以看到，当位置1和3同时存在时，位置0才会影响到位置4。也就是说：当左边和上边都存在时，左上角才有影响力。同理可得：**当左边不存在时，左上角和左下角都将失去影响力。**（注意这点，后面就是靠这条理论来列举情况的）

现在来列举所有情况。可以不考虑对角，只考虑边的组合。4边的组合有16种，可以分为5类：

1. 4边都没有（1种）

    此时对角全失去影响力。位置4只有1种情况，即情况编号46

2. 只有1边（4种）

    此时对角依然全失去影响力。位置4只有4种情况，即情况42,43,44,45

3. 有2边（6种）

    3.1 对边情况（2种）

    即左边和右边同时出现，或者上边和下边同时出现，此时对角依然全失去影响力。位置4只有2种情况，即情况32,33

   3.2 临边情况（4种）

   考虑左边和上边同时出现，此时左上角有影响力，有左上角出现与不出现2种情况。其他临边情况一样。位置4共有4\*2=8种情况，即情况34-41

4. 有3边（4种）

    考虑左上右同时出现，此时左上角和右上角有影响力，2个角有4种出现情况。所以共有4\*4=16种情况。即情况16-31

5. 4边都有（1种）

    此时所有角都有影响力，4个角有16种出现情况。所以共有1\*16=16种情况。即情况0-15

上面的解析有点绕口，大家可以参考图4理解一下。

所有情况加起来刚好47种。

分析好了所有情况，就可以列出每一种情况对应的哪4个小元件

下面把我辛苦写出来的对应表给出：

等号左边为情况编号，右边为小元件表，注意小元件表里面小元件的顺序，是以

“左上，右上，左下，右下”的顺序的。

[0] = [26,27,32,33]

[1] = [4,27,32,33]

[2] = [26,5,32,33]

[3] = [4,5,32,33]

[4] = [26,27,32,11]

[5] = [4,27,32,11]

[6] = [26,5,32,11]

[7] = [4,5,32,11]

[8] = [26,27,10,33]

[9] = [4,27,10,33]

[10] = [26,5,10,33]

[11] = [4,5,10,33]

[12] = [26,27,10,11]

[13] = [4,27,10,11]

[14] = [26,5,10,11]

[15] = [4,5,10,11]

[16] = [24,25,30,31]

[17] = [24,5,30,31]

[18] = [24,25,30,11]

[19] = [24,5,30,11]

[20] = [14,15,20,21]

[21] = [14,15,20,11]

[22] = [14,15,10,21]

[23] = [14,15,10,11]

[24] = [28,29,34,35]

[25] = [28,29,10,35]

[26] = [4,29,34,35]

[27] = [4,29,10,35]

[28] = [26,27,44,45]

[29] = [4,39,44,45]

[30] = [38,5,44,45]

[31] = [4,5,44,45]

[32] = [24,29,30,35]

[33] = [14,15,44,45]

[34] = [12,13,18,19]

[35] = [12,13,18,11]

[36] = [16,17,22,23]

[37] = [16,17,10,23]

[38] = [40,41,46,47]

[39] = [4,41,46,47]

[40] = [36,37,42,43]

[41] = [36,5,42,43]

[42] = [12,17,18,23]

[43] = [12,13,42,43]

[44] = [36,41,42,47]

[45] = [16,17,46,47]

[46] = [12,17,42,47]

[47] = [12,17,42,47]

**3.绘制**

绘制的过程大概是以下伪代码

1. // 在row行，col列绘制一个自动地图元件
2. **void** **drawAt**(**int** row, **int** col)
3. {
4. // 判断row和col是否越界
6. // 将[row,col]位置填为47号情况
7. // 这里可以填任意情况，只要确保[row,col]位置不为空即可
8. drawTileIndex(row,col,47);
10. // 更新[row,col]以及其周围8个格子的状态
11. updateTileState(row,col);
12. updateTileState(row-1,col-1);
13. updateTileState(row-1,col);
14. updateTileState(row-1,col+1);
15. updateTileState(row,col-1);
16. updateTileState(row,col+1);
17. updateTileState(row+1,col-1);
18. updateTileState(row+1,col);
19. updateTileState(row+1,col+1);
20. }
21. **void** **drawTileIndex**(**int** row, **int** col, **int** index)
22. {
23. // 根据index得到对应的小元件表
24. // 根据小元件表拼接成地图元件
25. }
26. **void** **updateTileState**(**int** row, **int** col)
27. {
28. // 如果该位置没有地图元件，则直接返回
29. **if** (!hasTileAt(row,col)) **return**;
31. // 判断其周围8个格子的状态state
32. // 根据判断的状态确定情况的编号index
33. drawTileIndex(row,col,index);
34. }

这里最麻烦的就是“根据判断的状态确定情况的编号index”

由于情况有47种之多，而且判断的过程也很纠结。这里提供我的方法，虽然也很麻烦，不过逻辑比较清晰。

核心就是用二进制表达，边角共有8个，用一个8位的char即可。首先定义如下:4边4角各占一位

1. left = 0000 0001
2. right = 0000 0010
3. top = 0000 0100
4. bottom = 0000 1000
5. left\_top = 0001 0000
6. right\_top = 0010 0000
7. left\_bottom = 0100 0000
8. right\_bottom = 1000 0000

那么当前4边4角的情况就可以上面的组合出来。

然后就可以用switch case表达47种情况，如

1. **switch**(state)
2. {
3. **case** 1111 1111: /\*即4边4角都有\*/ index = 0; **break**;
4. **case** 1100 1000: /\*即左边和上边以及左上角都有\*/ index = 38; **break**;
5. ...
6. }

**4.小结**

没想到写这样一篇文章会这么费力，在写这家伙时，我总感觉自己的方法太笨了。我感觉会有好的方法的，所以还是坚持写下来，抛砖引玉了。