Cocos2dx 数据结构，数据存储和TileMap

目 录 / contents

section1

数据结构

Vector

Map

Value

数据结构是计算机存储，组织数据的方式，适当使用数据结构让编程事半功倍，游戏过程实则上是用户和数据之间的交互，如何合理的组织和存储游戏中的各种元素，cocos2dx为我们提供了有效的解决方案。

cocos2d::Vector<T>

------类比c++ STL中的vector<T>

- cocos2d::Vector<T>是一个封装好的动态增长的顺序访问的容器。
- 元素是按序存取的，其底层实现数据结构是标准模版库STL中的顺序容器std::vector。
- 时间复杂度：
  - 随机访问，O(1)；
  - 将元素插入到尾部或删除尾部元素，O(1)；
  - 随机插入或删除, O(n)

模版参数

- T的类型必须是继承自cocos2d::Object类型的指针。因为已经将Cocos2d-x的内存管理模型集成到了cocos2d::Vector<T>中，所以类型参数不能是其他的类型包括基本类型。

内存管理

- cocos2d::Vector<T>类只包含一个成员数据：std::vector<T> _data;
- _data的内存管理是由编译器自动处理的，如果声明了一个cocos2d::Vector<T>类型，就不必费心去释放内存。 注意：使用现代的c++，本地存储对象比堆存储对象好。所以请不要用new操作来申请cocos2d::Vector<T>的堆对象，请使用栈对象。

基本用法

- 警告：cocos2d::Vector<T>并没有重载[]操作，所以不能直接用下标[i]来获取第i位元素。 cocos2d::Vector<T>提供了不同类型的迭代器，所以我们可以受益于c++的标准函数库，我们可以使用大量标准泛型算法和for_each循环。 除了std::vector容器的操作之外，开发者们还加入许多标准算法诸如：std::find, std::reverse和std::swap，这些算法可以简化很多通用的操作。

简易使用方法及例子

```
//创建精灵对象
Sprite* sprite = Sprite::create("CloseNormal.png");

//在栈上申请Vector：
      cocos2d::Vector<Sprite*>  container;

//在数组的最后插入一个对象指针

      container.pushback(sprite);
//  在数组位置1插入一个对象指针
      container.insert(1,sprite);
```

简易使用方法及例子

```
//  在数组位置1插入一个对象指针
    container.insert(1,sprite);

//  判断对象是否在容器内  返回bool值
    bool  isHere = container.contains(sprite);

//获取位置为1的对象指针
    Sprite* pSprite = container.at(1);
```

如何遍历数组

```
//简单的遍历，使用c++11 的auto语法
for(auto sp : container)
{
    //do something
}

//利用迭代器遍历
cocos2d:Vector<Sprite*>::iterator it = container.begin();
for(;it != container.end();)
{
        //(*it)->function();
}
```

一种在遍历中删除数组元素的方法

```
cocos2d:Vector<Sprite*>::iterator it = container.begin();
for(;it != container.end();){
        if (sprite_1==(*it)){
        //erase()执行后会返回指向下一个元素的迭代器
                        it = container.erase(it);
                }
                else{
                        it++;
                        //do something
                }
}
```

cocos2d::Map<K,V>

- cocos2d::Map<K,V>是使用std::unordered_map作为底层结构的关联式容器。 而std::unordered_map是一个存储键值对的关联式容器，它可以通过它们的键快速检索对应的值。 使用unordered_map，键通常是唯一的，而值则与这个键对应。在unordered_map内部，元素是无序，它们是根据键的哈希值来存取的，存取的时间复杂度是常量，超级快。

模版参数

- cocos2d::Map<K,V>类只包含一个数据成员：
- typedef std::unordered_map<K, V> RefMap;
- RefMap _data;
- _data的内存管理是由编译器处理的，当在栈中声明cocos2d::Map<K,V>对象时，无需费心释放它占用的内存。

基本用例

- cocos2d::Map<K,V>并没有重载[]操作，不要用下标[i]来取cocos2d::Map<K,V>对象中的元素。

简易使用方法及例子

```
//创建精灵对象
Sprite* sprite = Sprite::create("CloseNormal.png");

//在栈上申请Map：
        cocos2d::Map<std::string,Sprite*>   map;

//插入键值对
        map.insert("monster",sprite);

//返回map中key映射的元素的值
        map.at("monster");
```

cocos2d::Value

- cocos2d::Value是许多基本类型的封装
  - (int,float,double,bool,unsigned char,char*和std::string)
  - std::vector<Value>, std::unordered_map<std::string,Value>和 std::unordered_map<int,Value>
- 可以将上面提及的基本类型与cocos2d::Value类互相转换。
- cocos2d::Value底层用一个统一的变量来保存任意基本类型值，它将更加节省内存

简易使用方法及例子

```
//利用默认构造器创建
Value val;

//用字符串初始化
Value val("hello");

//用整型数初始化
Value val(14);

//.....
```

section2

本地数据存储

UserDefault

SQLite

一个游戏中，有经常处于变化的动态数据，亦有不常变动，相对稳定的静态数据，例如一个游戏环境的设置状态（音乐开关，音量大小等），还有关卡的通关数，最高分记录等，通常会把这些数据存储在一个本地数据文件中，游戏运行时可以随时进行加载或更改。

UserDefault

- 最简单的数据存储类，一个灵巧方便的微型数据库
- 适用于基础数据类型的存取
- 数据将以xml文件格式存储

常用使用方法

```
#define database UserDefault::getInstance()

//检测xml文件是否存在（非必须）
if (!database->getBoolForKey("isExist")){
        database->setBoolForKey("isExist", true);
    }

// 简单存取
    int value = 14;
    database->setIntegerForKey("value", value);
    database->setStringForKey("string", "hello");
```

常用使用方法

执行上述代码后，会在本地计算机的某个地方生成一个userdefault.xml文件，请同学们阅读源码，想办法找出文件所在位置并查看里面的内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<userDefaultRoot>
    <isExist>true</isExist>
    <value>14</value>
    <string>hello</string>
</userDefaultRoot>
```

SQLite

- 使用非常广泛的嵌入式数据库，它有小巧 、高效、跨平台、开源免费和易操作的特点。
- 十分适用于移动游戏应用的开发。

SQLite是一个软件库，实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。SQLite是一个增长最快的数据库引擎，这是在普及方面的增长，与它的尺寸大小无关。SQLite 源代码不受版权限制。

*SQLite学习网站：http://www.w3cschool.cc/sqlite/sqlite-tutorial.html

创建数据库

```
//包含相关文件
#include "sqlite3.h"
//数据库指针
sqlite3* pdb =NULL
//数据库路径
string path= FileUtils::getInstance()->getWritablePath()+"save.db"
//根据路径path打开或创建数据库
int result = sqlite3_open(path.c_str(),&pdb);
//若成功result等于SQLITE_OK
```

创建新表

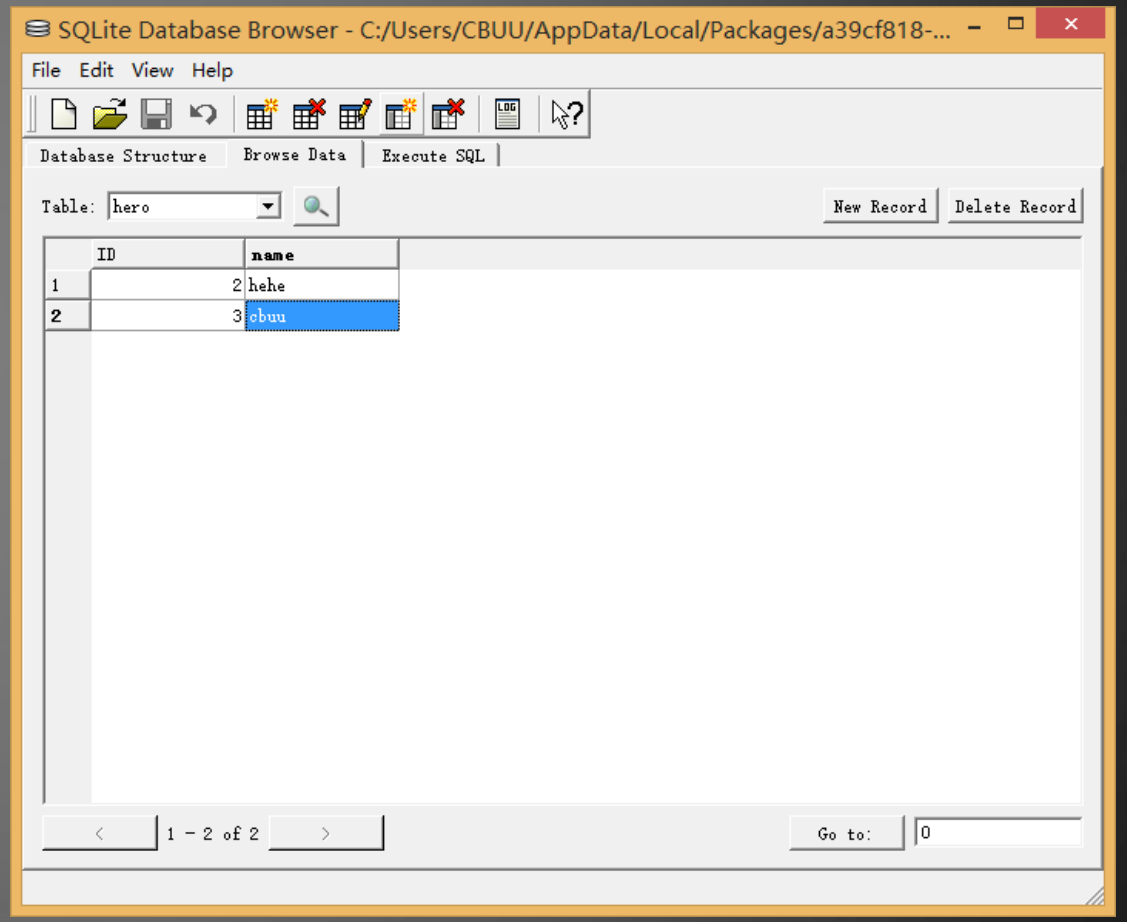
```
//SQLite语句 创建一个主键为ID名字为hero的表
std::string sql = "create table hero(ID int primary key not
null,name char(10));";
/*运行SQLite语句，运行参数分别为数据库指针，SQLite语句，回调函数，
回调参数，错误信息 */
        result = sqlite3_exec(pdb, sql.c_str(), NULL,NULL,NULL);
```

SQLite语句 增删改查

```
//向表中插入一条 ID为１， name为iori 的数据
sql = "insert into hero values(1,'iori');";

//删除表中id为1的一条数据
sql = "delete from hero where id=1;";

//把id为2的数据name改为hehe
sql = "update hero set name='hehe' where id=2;";
```

```
char **re;//查询结果

int row, col;//行、列

//根据语句获取表中数据
sqlite3_get_table(pdb, "select * from hero", &re, &row,
&col, NULL);

//遍历存储数组打印数据
for (int i = 1; i <= row; i++)//2{
        for (int j = 0; j < col; j++){
                log("%s", re[i*col + j]);
        }
}

//查询后注意释放指针
sqlite3_free_table(re);
```

运行以上语句后，应用会创建一个database.db数据库文件，
你可以找到此文件并打开查看甚至修改内容。



*推荐一款超轻量级sqlite数据库文件可视化软件：

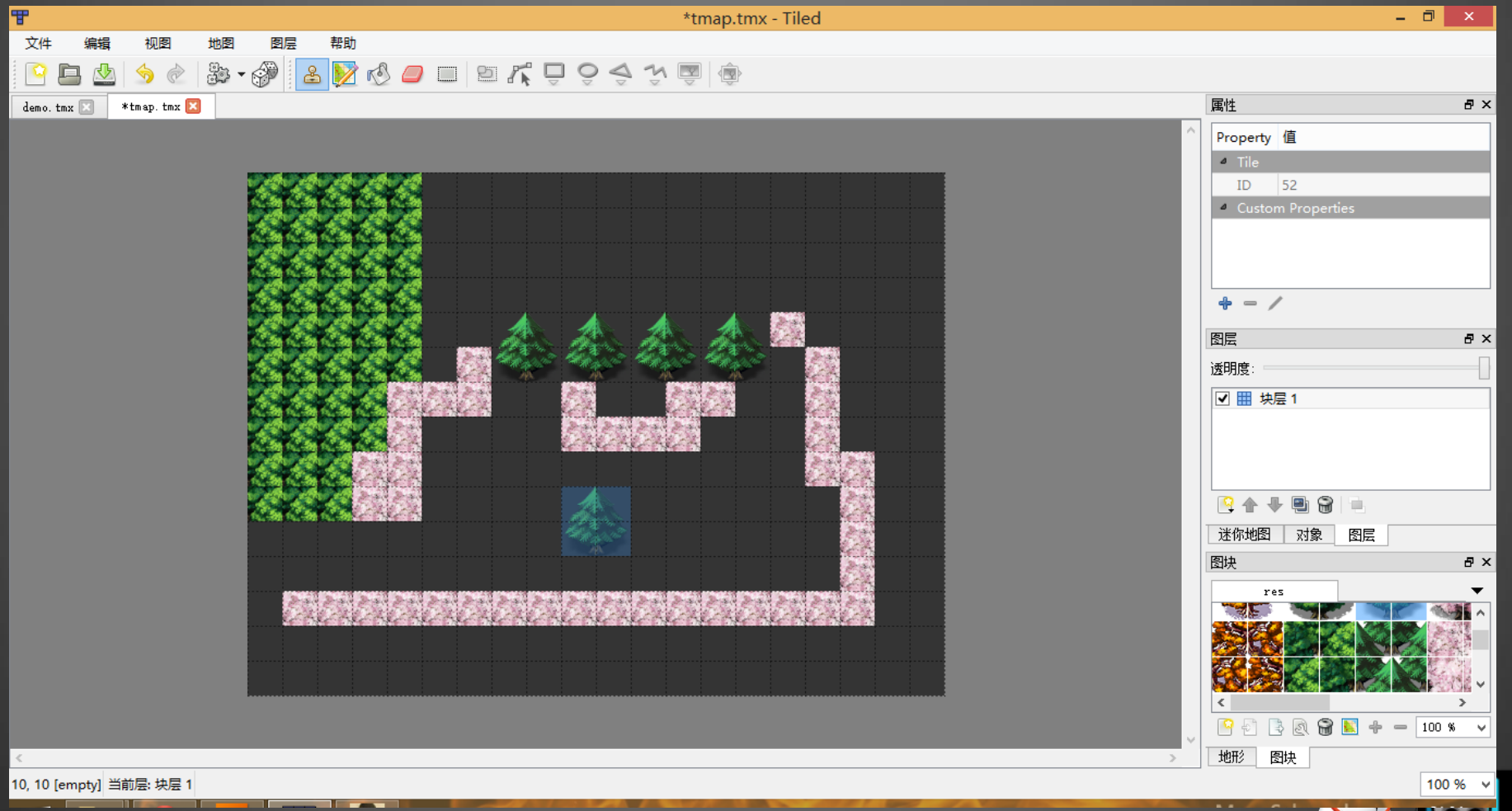
SQLite Datebase browser

section3

瓦片地图与TileMap的使用

TileMap:

一张大的世界地图或者背景图可以由几种地形来表示，每种地形对应一张小的的图片，我们称这些小的地形图片为瓦片（图块）。把这些瓦片拼接在一起，一个完整的地图就组合出来了

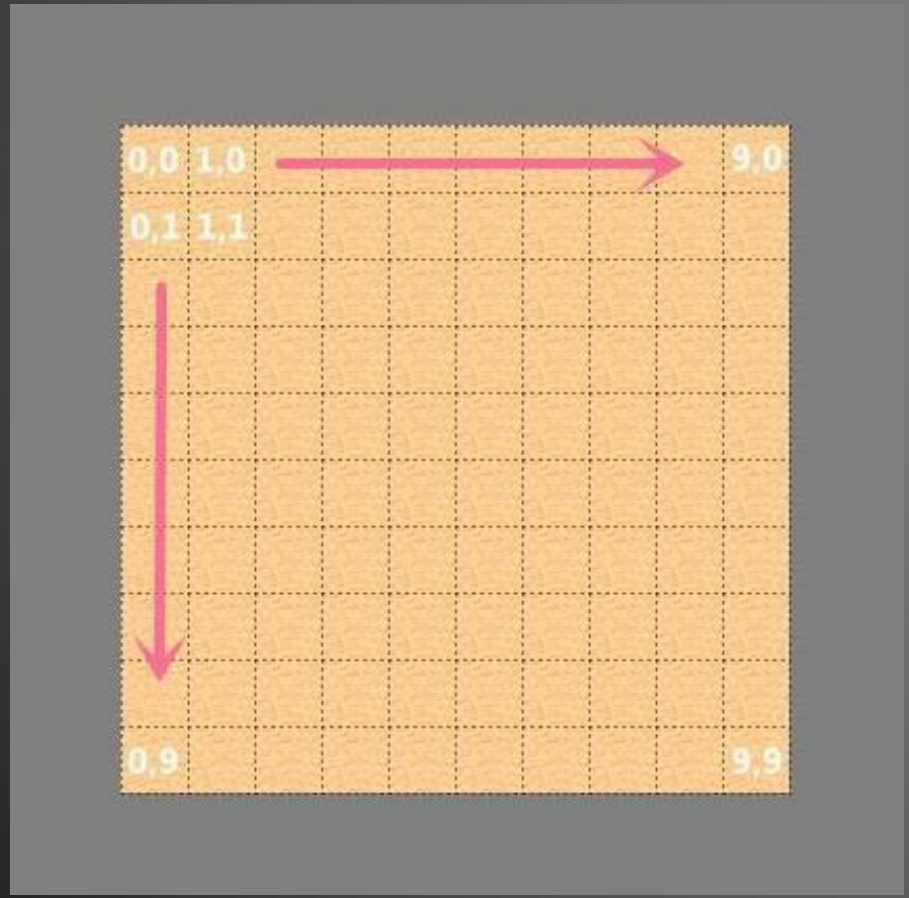


Tilemap的地图方向和坐标系:

地图编辑器可以制作三类地图：普通地图（直90°）、 斜45°地图、 斜45°交错地图。 除此之外，而Cocos引擎还支
持六边形地图
瓦片地图的坐标系原点在左上角，x轴从左到右，y轴从上到下

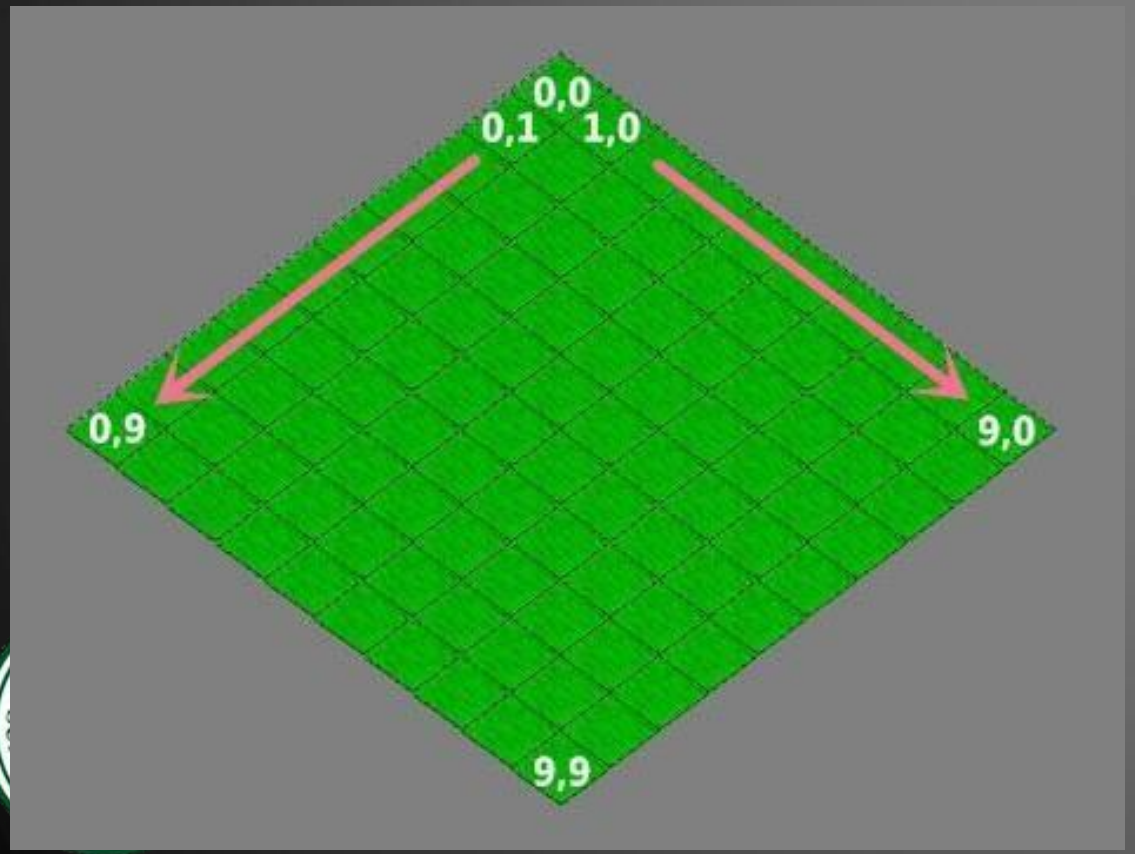
普通地图（直90°）

一般用于制作2d游戏

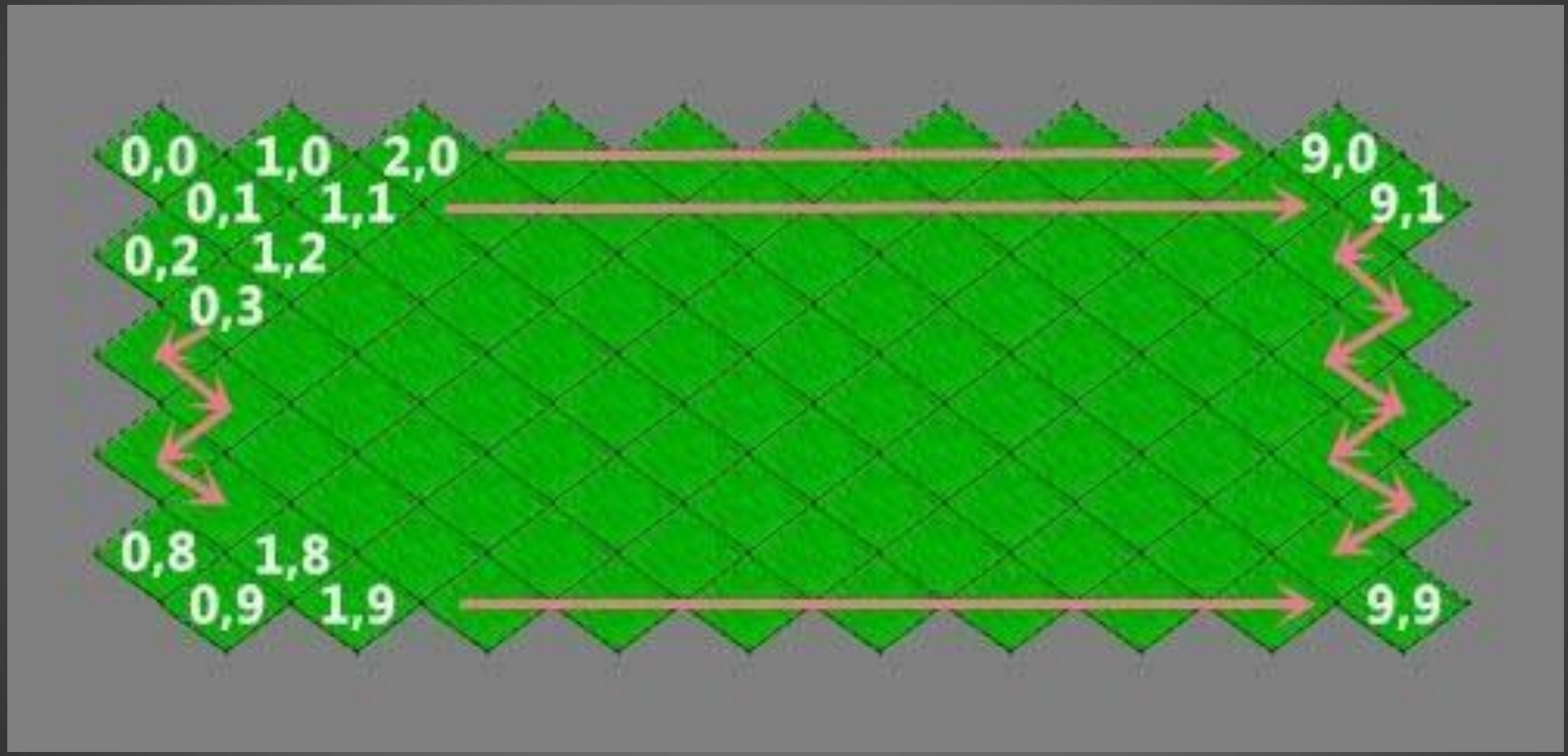


斜45° 地图

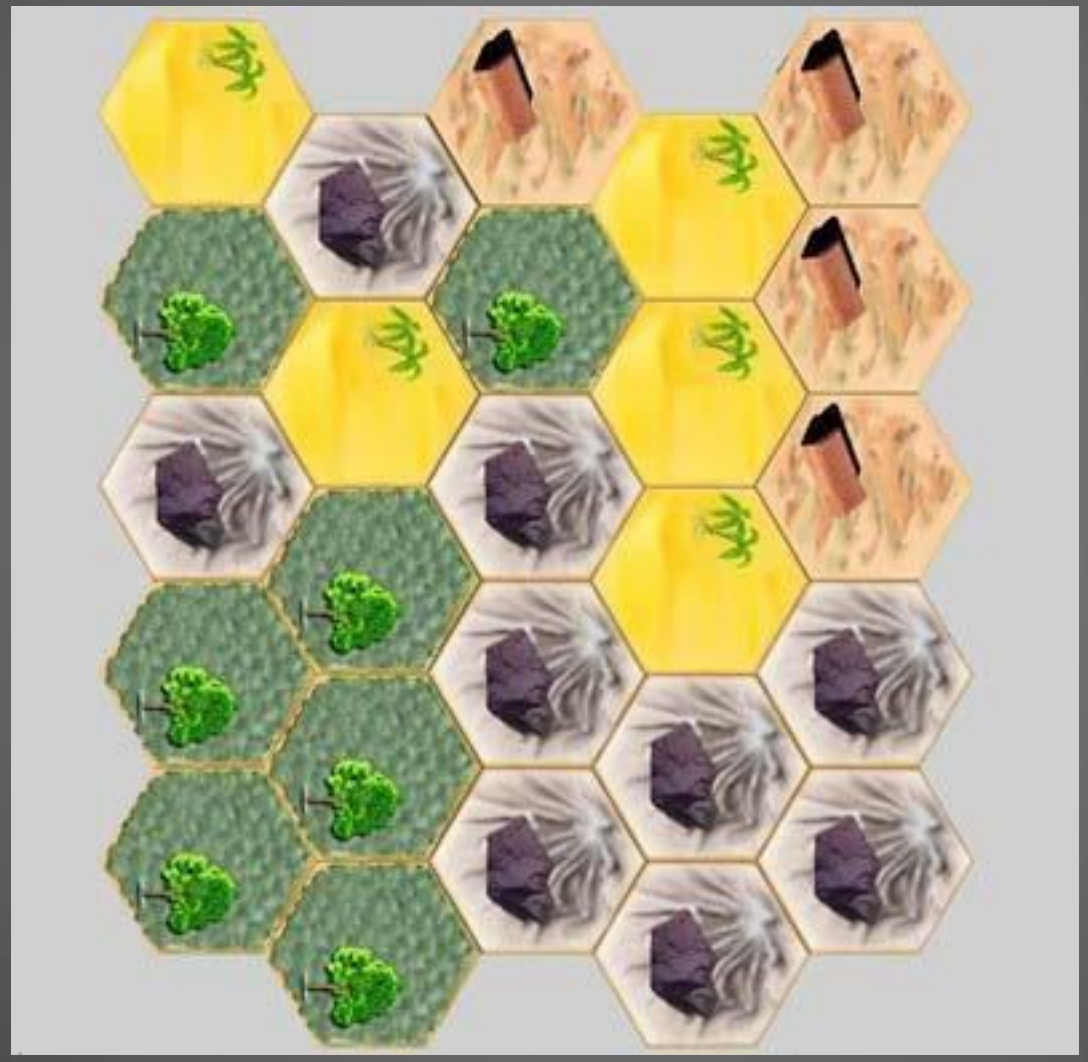
45°的tilemap具有伪3d的效果



斜45° 交错地图



支持六边形地图



Tilemap的地图层:

（1）每一个地图层可以被表示为TMXLayer类，并设置了名称。（如下图有三个地图层：Meta、 Foreground、 BackGround ）。

（2）每一个单一的瓦片被表示为Sprite类，父节点为TMXLayer。

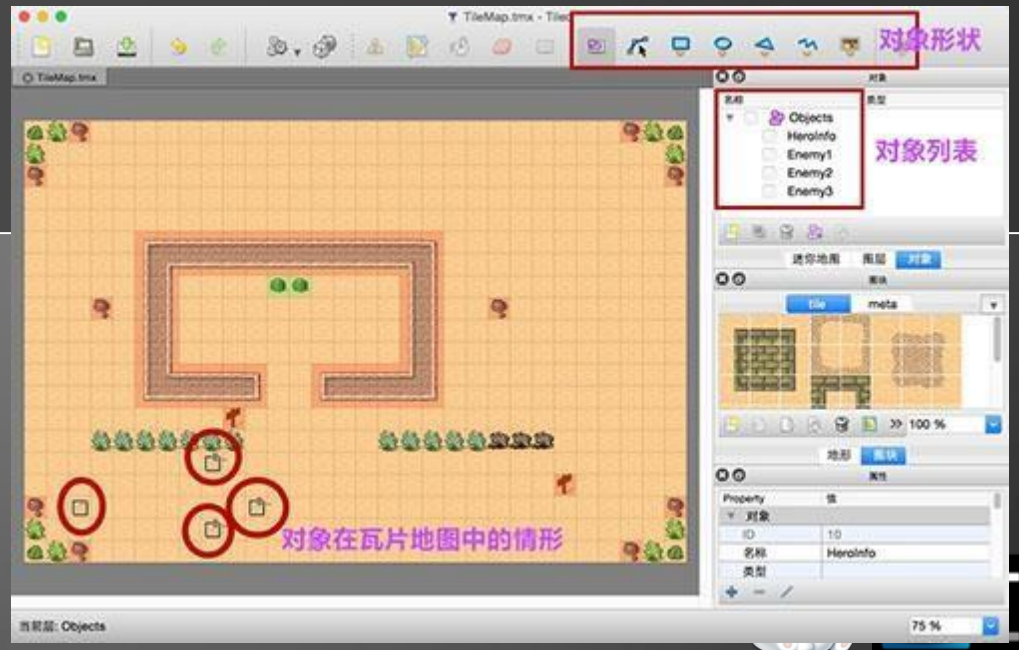
（3）每一个地图层只能由一套瓦片素材组成，否则会出问题。

Tilemap的对象层:

（1）用来添加除背景以外的游戏元素信息，如道具、障碍物等对象。

（2）一个对象层可以添加多个对象，每个对象的区域形状的单位是：像素点。

（3）对象层中的对象在TMX文件中以 键值对（key-value）形式存在，因此可以直接在TMX文件中对其进行修改。



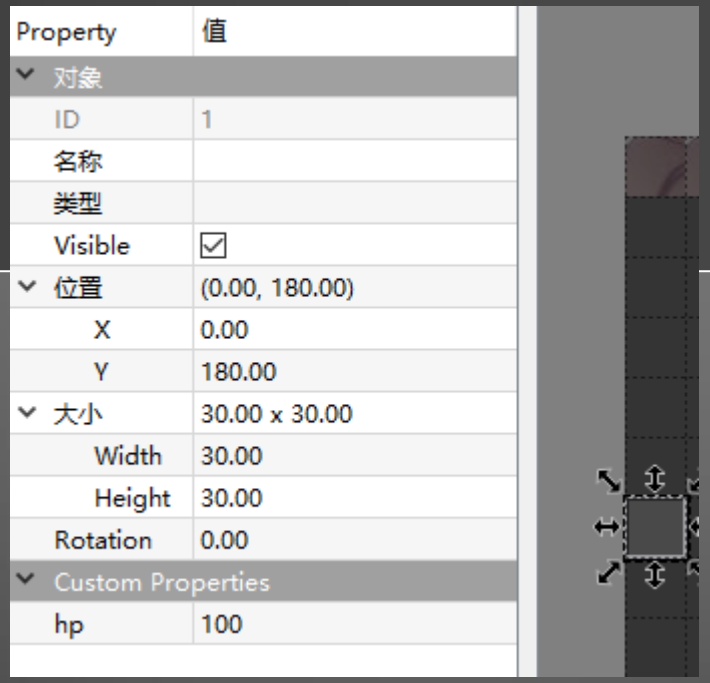
Tilemap瓦片的全局标识GID:

在Cocos游戏中，每一个瓦片素材都有一个全局唯一标识GID，而瓦片的GID就是表示该瓦片所使用的是哪个GID的图块素材。

Tilemap瓦片地图的属性值:

自定义的属性可以在地图编辑器中进行设置，并且可以在代码中获取这些属性以及对应的属性值。

只要点击“目标”，就可以看到它的属性，并且可以添加自定义属性（Custom Properties）。

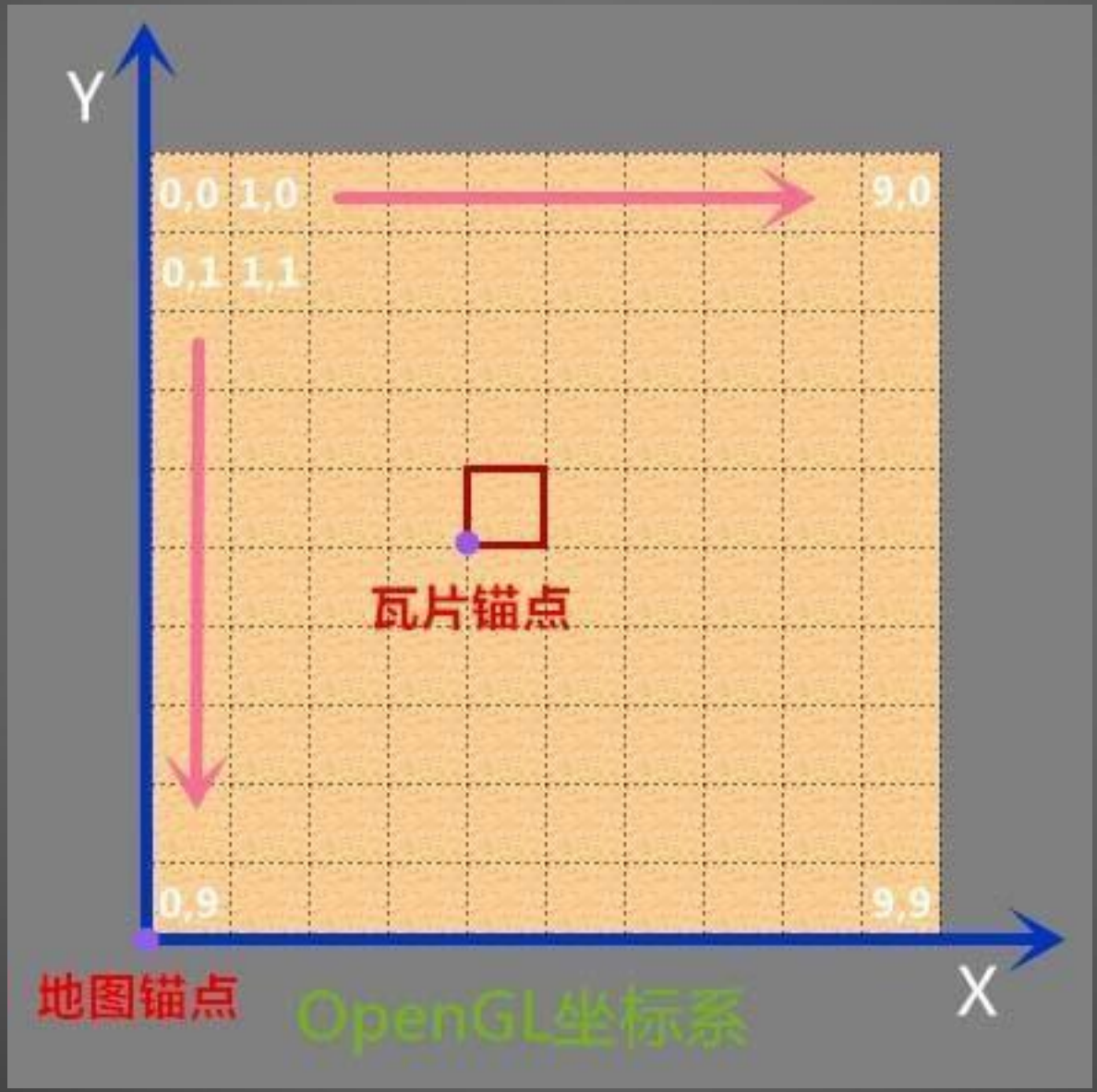


瓦片地图的锚点：

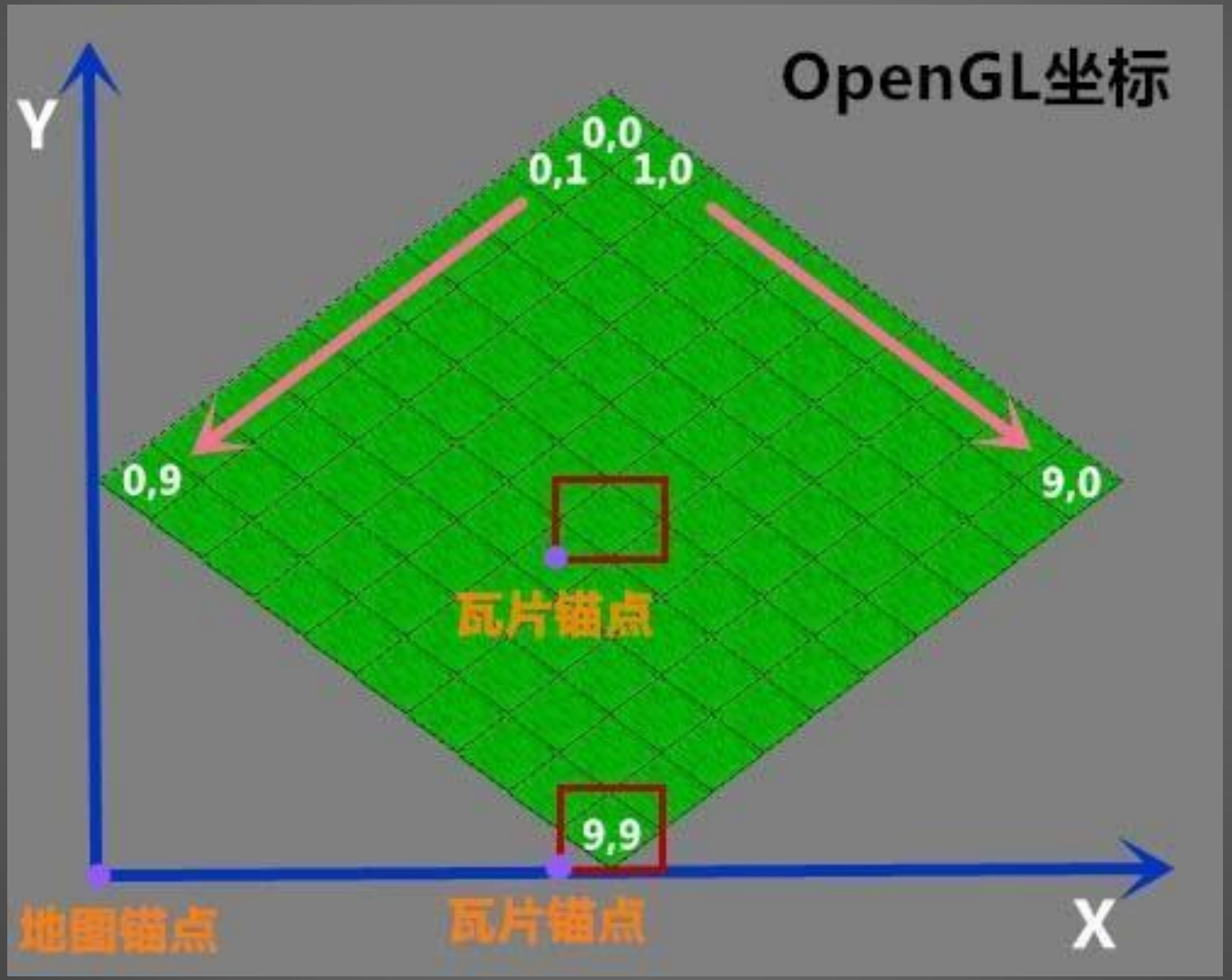
瓦片地图的锚点默认为（ 0，0），每个瓦片的锚点默认也为（0，0）。

PS：锚点是可以设置的，因为它不是继承于Layer，而是直接继承于Node。

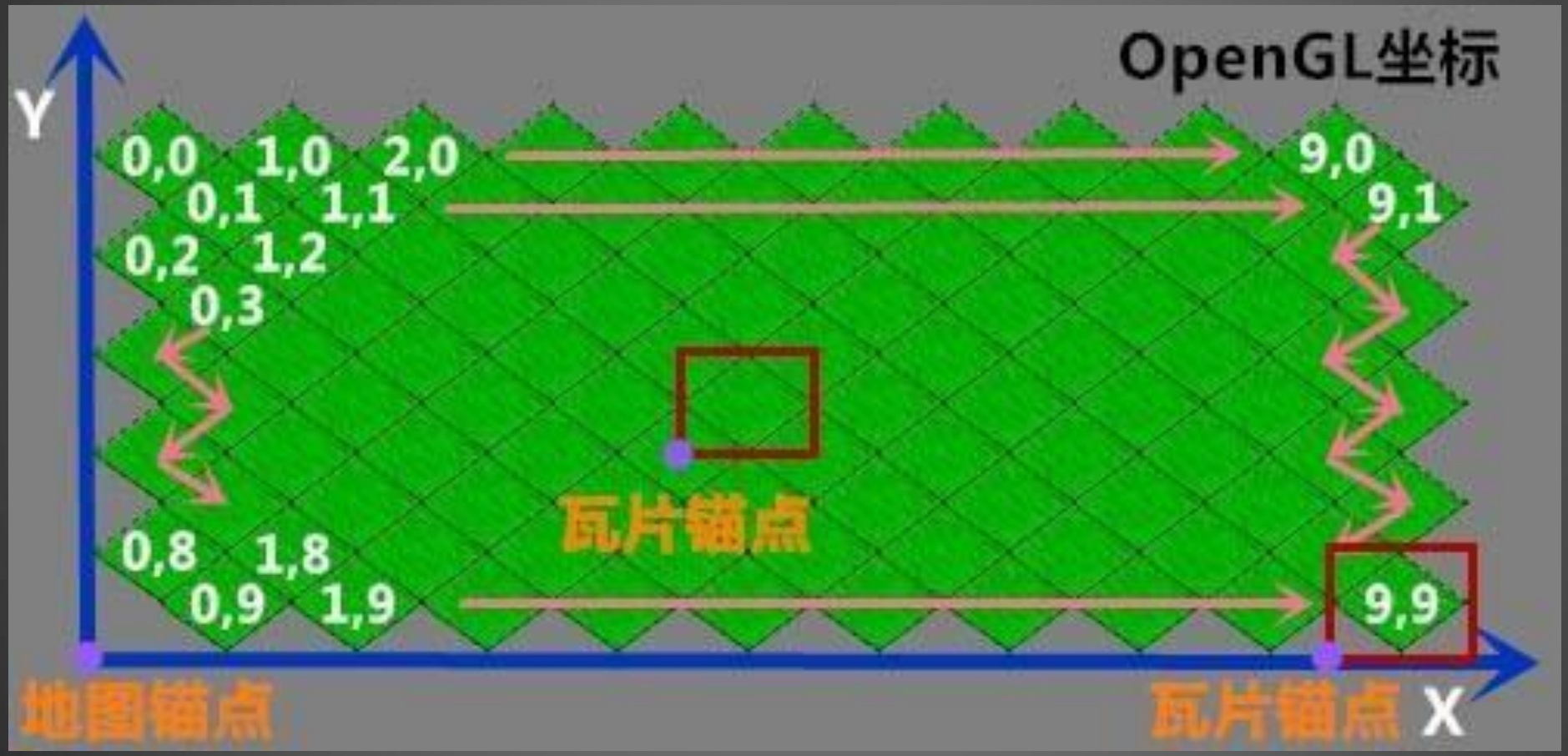
普通瓦片地图的锚点：



45度瓦片地图的锚点：



斜45°交错瓦片地图的锚点：



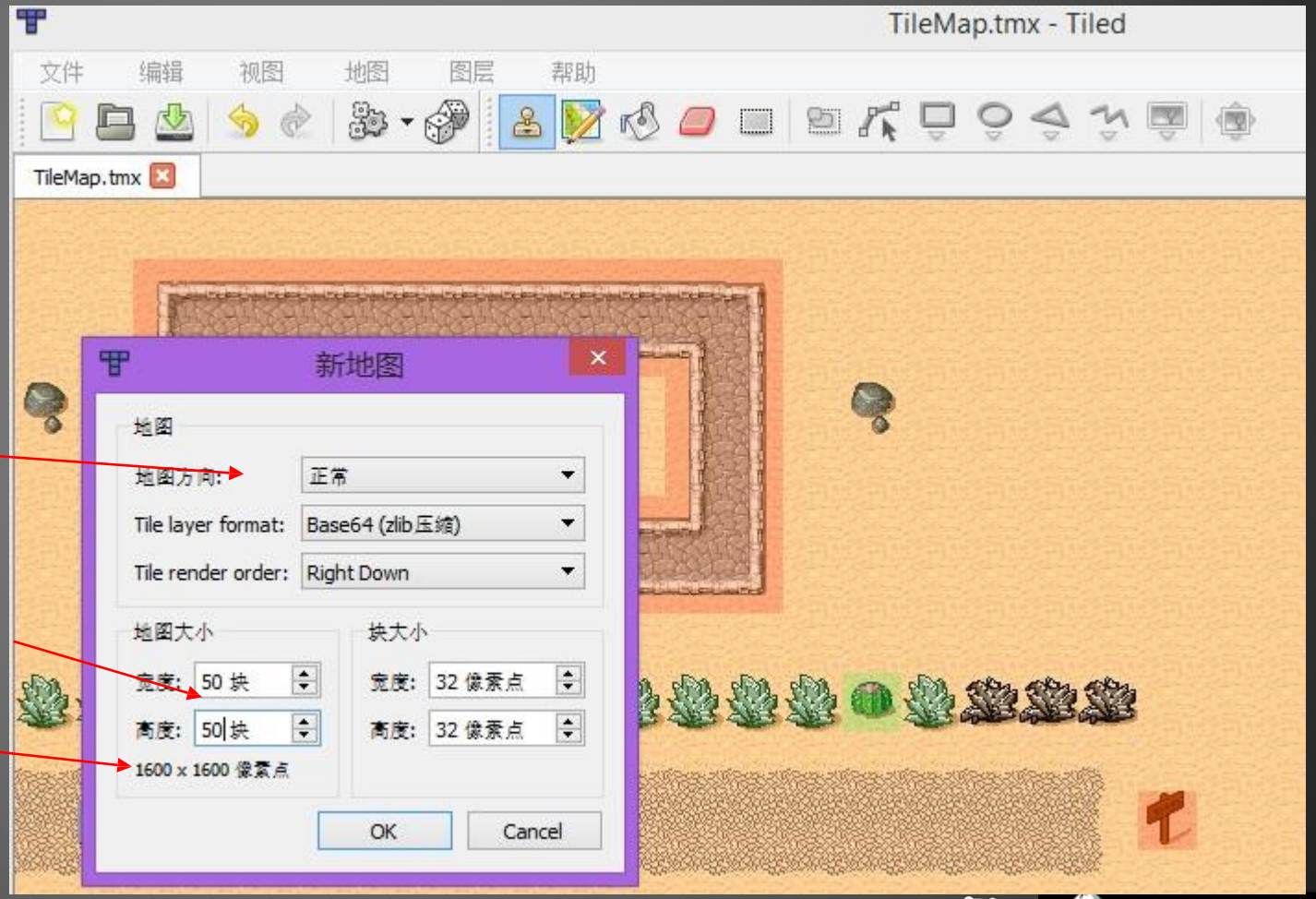
地图层之间的遮罩关系：

每个地图层的 zOrder（渲染顺序）会根据在地图编辑器中设置的前后关系进行设置。由下往上设置 zOrder 值，最靠后的 zOrder = 0，随后每个图层zOrder+1。

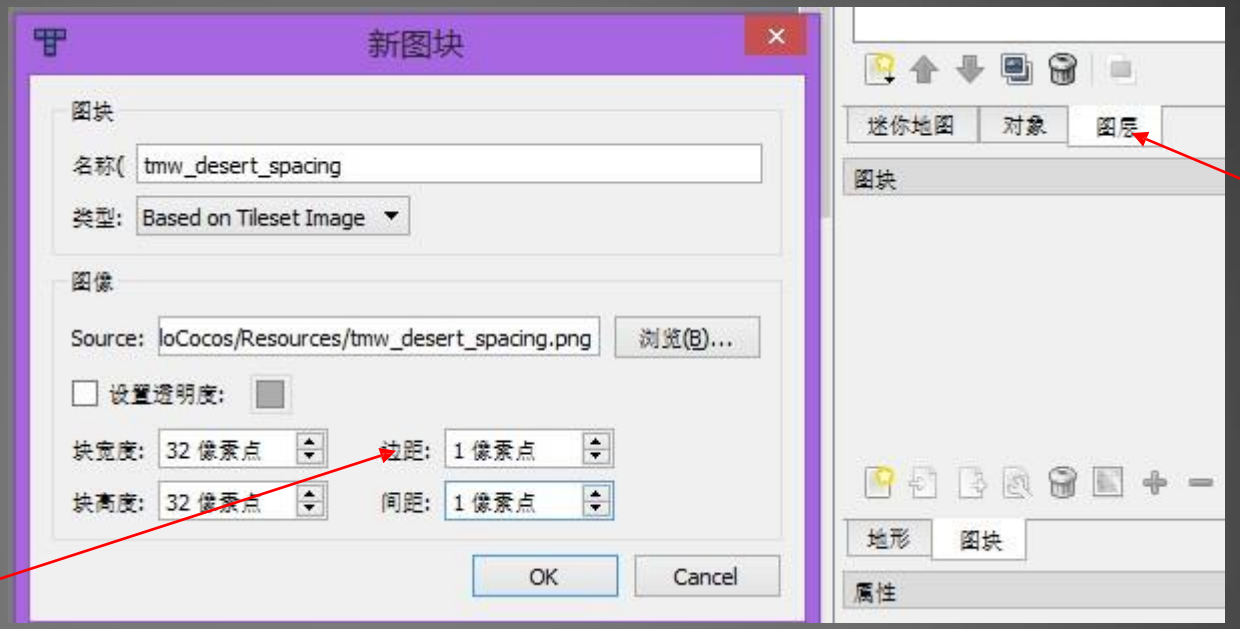
瓦片地图的制作与使用

打开TILEMAP，新建文件

- 方向可选择45度，45度交错等
- 这里的宽度与高度是指有多少“块”
- 这里才是你绘制的地图的大小



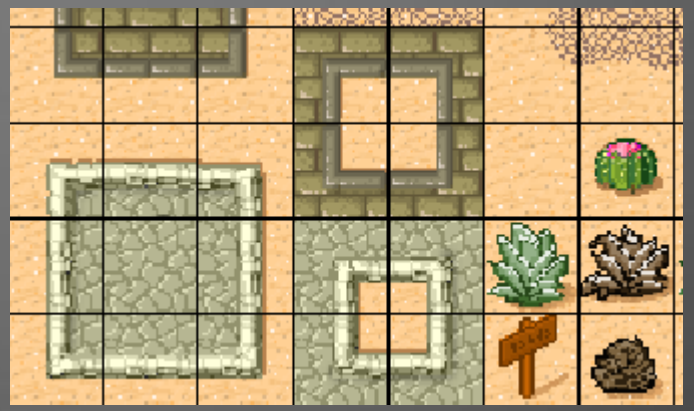
导入图块



- 新建图层

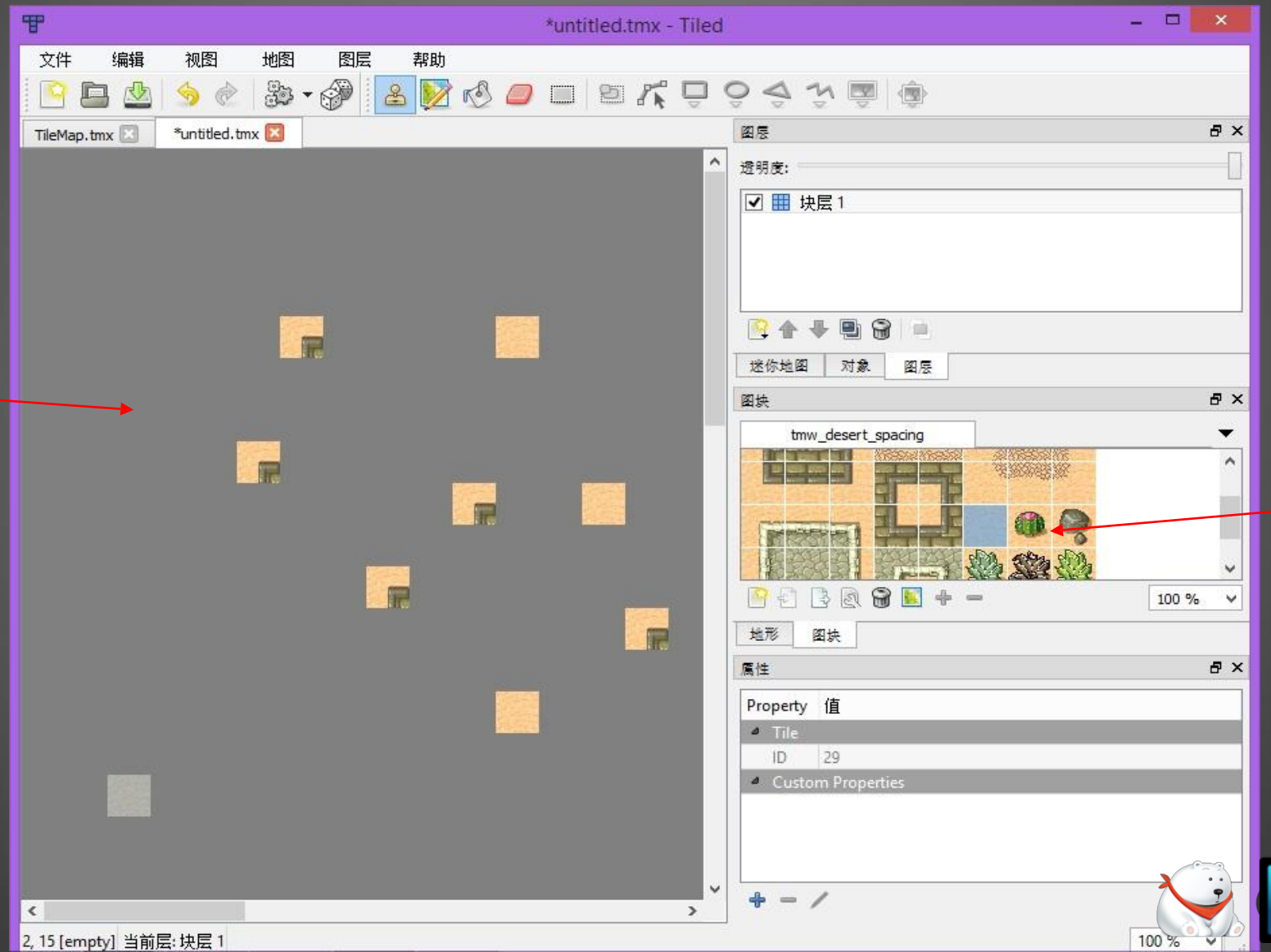
- 素材原因，这里边距，间距设为1个像素点

- 素材间有空白边界



绘制地图

- 任意摆放
- 选取图片

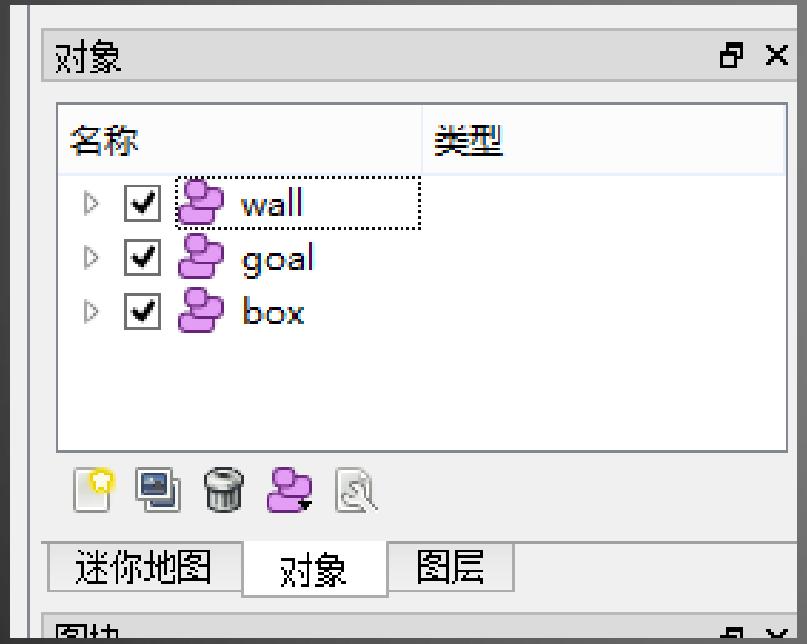


导入

```
//根据文件路径快速导入瓦片地图
TMXTiledMap* tmx = TMXTiledMap::create("map.tmx");

//设置位置
tmx->setPosition(visibleSize.width / 2, visibleSize.height / 2);

//设置锚点
tmx->setAnchorPoint(Vec2(0.5, 0.5));

//添加到游戏图层中，其中0代表Z轴（Z轴低的会被高的遮挡）
this->addChild(tmx,0);
```

对象层

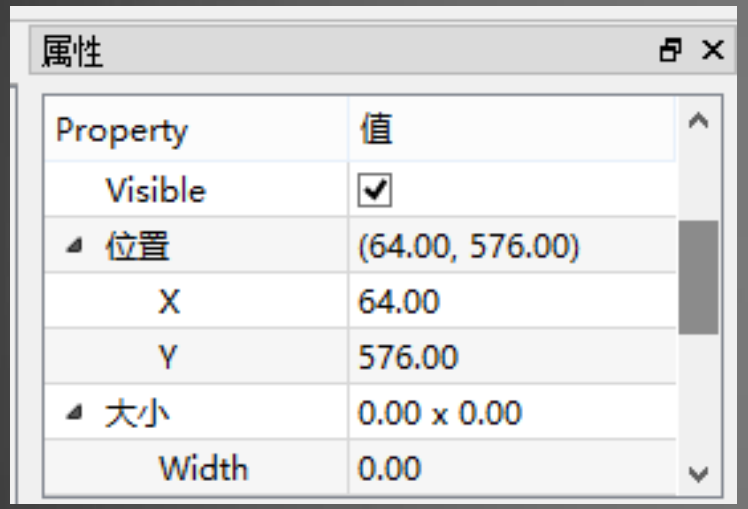
对象层允许你在地图上圈出一些区域，来指定一些事件的发生或放置一个游戏对象。比如，你想在地图放一堵墙来阻挡玩家前进，又或者设置一个开关，角色触碰后会触发某些事件。



解析对象层

```
//从tmx中获取对象层
TMXObjectGroup* objects = map->getObjectGroup(  “wall");

//从对象层中获取对象数组
ValueVector container = objects->getObjects();

//遍历对象
for(auto obj:container){
        ValueMap values = obj.asValueMap();
        //获取纵横轴坐标（cocos2dx坐标）
        int x = values.at("x").asInt()
        int y = values.at("y").asInt()
}
```

解析对象层




```
<object id="43" gid="1" x="64" y="384"/>
<object id="45" gid="1" x="192" y="384"/>
<object id="46" gid="1" x="320" y="384"/>
<object id="47" gid="1" x="384" y="384"/>
<object id="48" gid="1" x="448" y="384"/>
<object id="49" gid="1" x="512" y="384"/>
<object id="51" gid="1" x="256" y="512"/>
<object id="52" gid="1" x="128" y="256"/>
```

*如图所示，用文本编辑器打开tmx文件，发现对象属性以键值对的形式存储。

tilemap的参考教程：

http://www.cocos.com/doc/tutorial/show?id=2455

THE END

THANKS FOR WATCHING