Cocos2dx 游戏开发教程

by 邓立

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，游戏过程实则上是各种各样的游戏元素数据随着时间和用户的输入不断产生变化，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>

- 是一个封装好的能动态增长顺序访问的容器
- 低层实现数据结构是标准模版库中的标准顺序容器std::vector
- 注意：模板的类型 T 必须是继承自coco2dx::Ref的指针类型
- 注意：尽量不要使用new来申请Vector对象

Vector就是一个顺序存储的容器，是使用很普遍的数据结构，在游戏中可以用于方便地存储那些大量且普遍的元素。

例如在打飞机的游戏中，必须要有对敌机的存储与管理，方便我们对敌机进行操作以及清除。此时我们可以把所有敌机存储在一个Vector里。

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>

- 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。
- 注意：模板的类型 V 必须是继承自coco2dx::Ref的指针类型

Map是以键值对方式存储数据的关联容器，提供一对一的快速访问通道，就像查字典一样，通过目录的关键字符就可以迅速查询到想要的内容

（Map在旧的cocos2dx版本中对应于CCDictionary）

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

- 是对各种基本类型的一个包装类型
- 当要使用基本类型的聚合时，将基本类型包装成cocos2d::Value，然后将它们和模版容器cocos2d::Vector和cocos2d::Map联合使用（非必需）。

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。

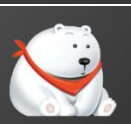
```
<?xml version="1.0" encoding="UTF-8"?>
<userDefaultRoot>
    <isExist>true</isExist>
    <value>14</value>
    <string>hello</string>
</userDefaultRoot>
```

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

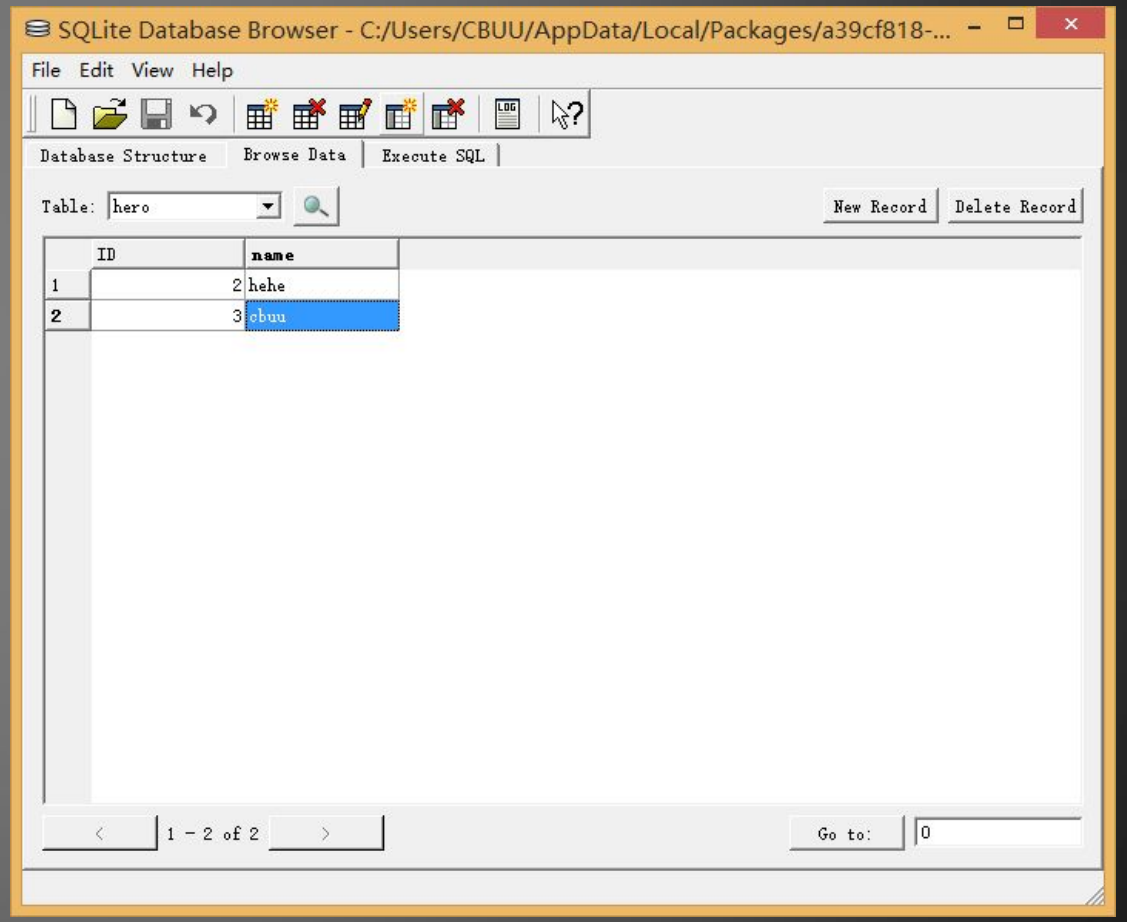
```
//SQLite语句 创建一个主键为ID名字为hero的表
std::string sql = "create table hero(ID int primary key not
null,name char(10));";
/*运行SQLite语句，运行参数分别为数据库指针，SQLite语句，回调函数，
回调参数，错误信息 */
        result = sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);
```

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：

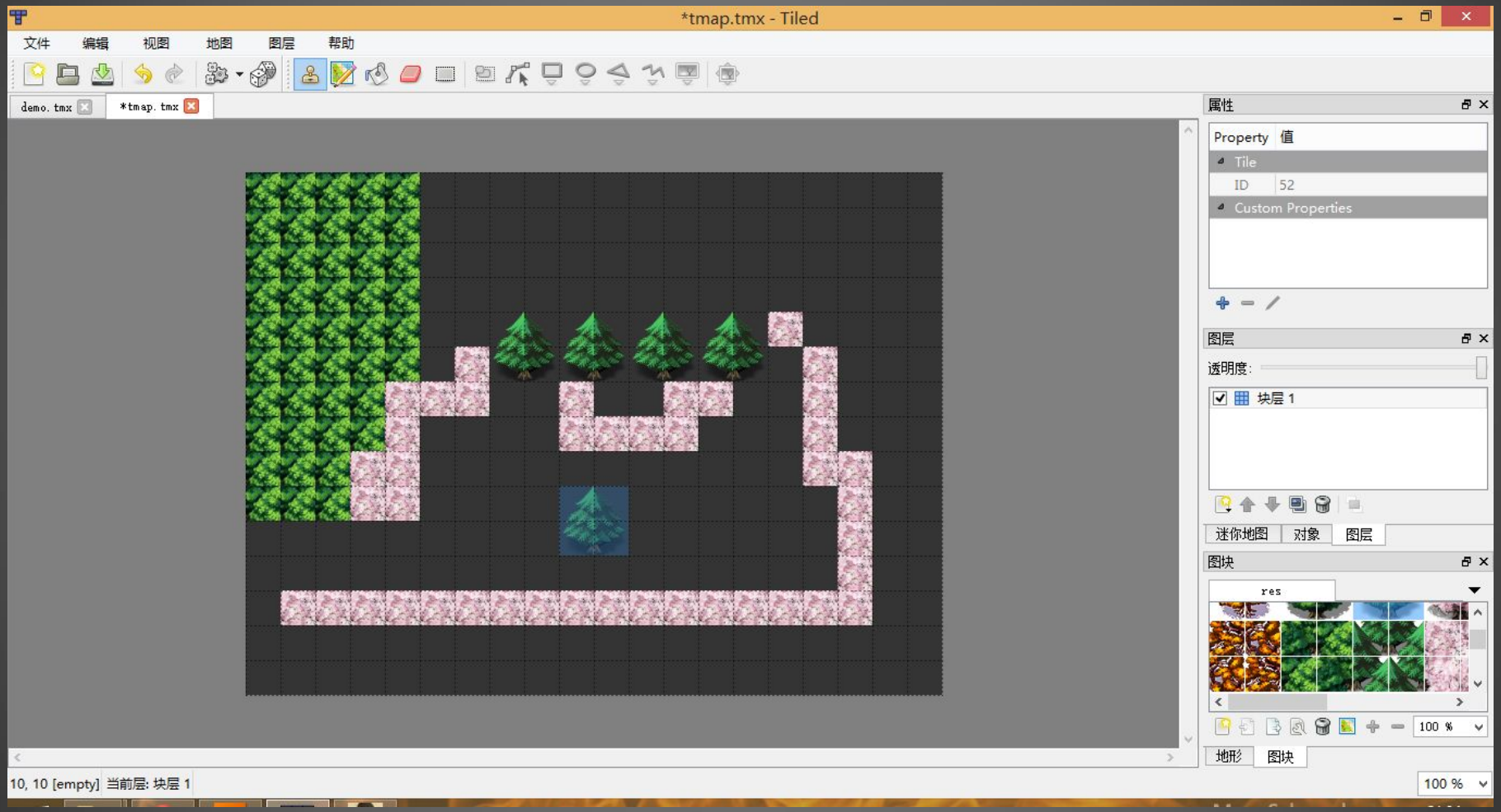
SQLite Datebase browser

section3

瓦片地图与tilemap使用

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

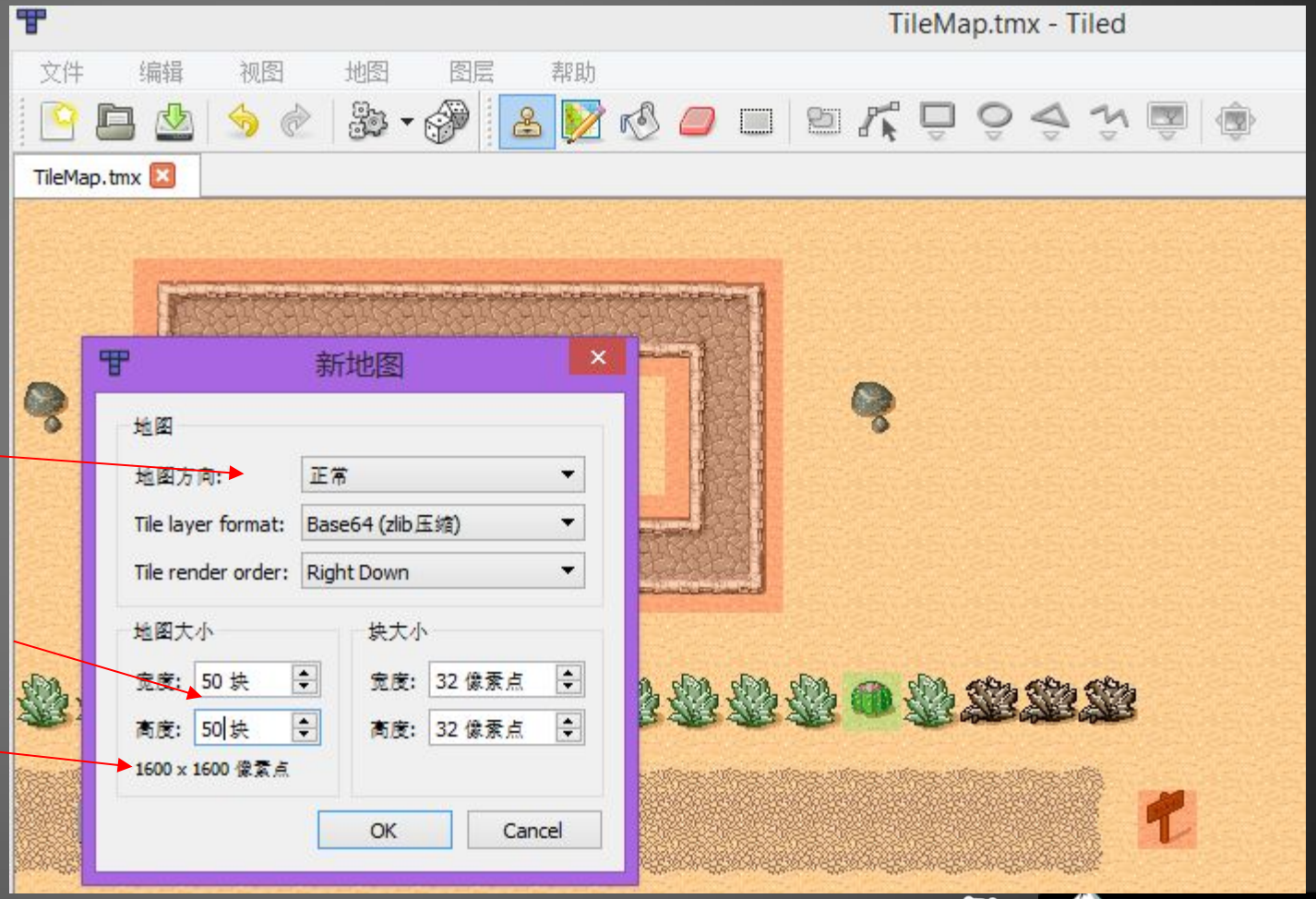
Tilemap:


*tmap.tmx - Tiled
文件
编辑
视图
地图
图层
帮助
demo. tmx
*tmap. tmx
属性
Property
值
Tile
ID
52
Custom Properties
图层
透明度:
块层 1
迷你地图
对象
图层
图块
res
100 %
地形
图块
10, 10 [empty] 当前层: 块层 1
100 %

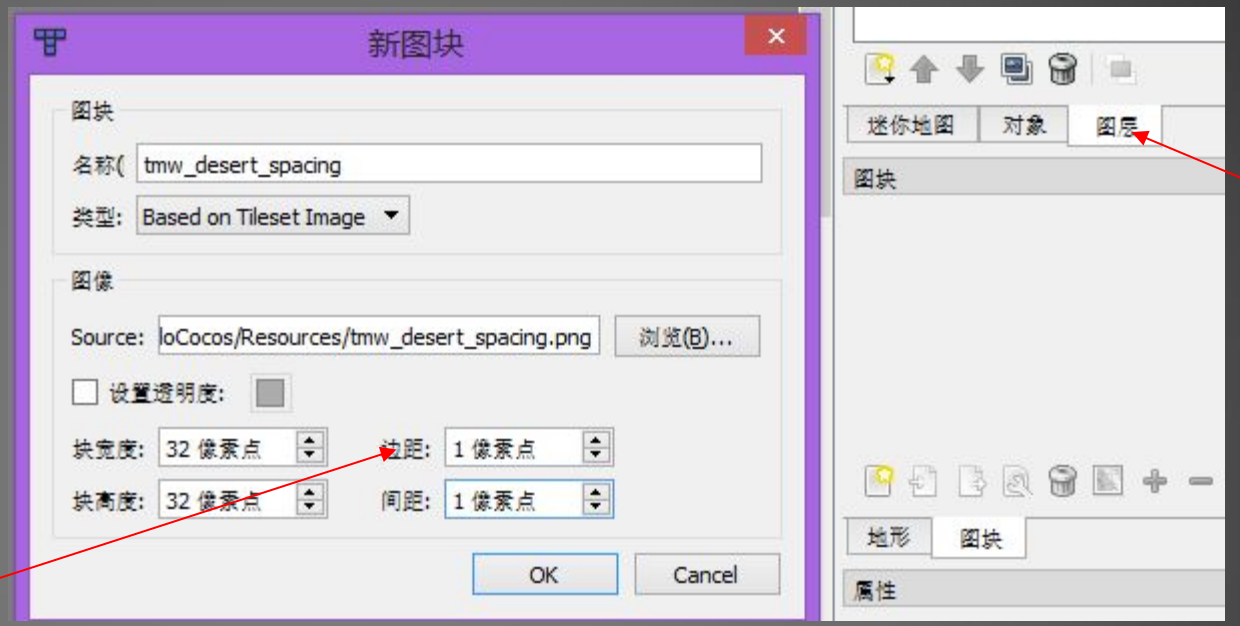
瓦片地图的制作与使用

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



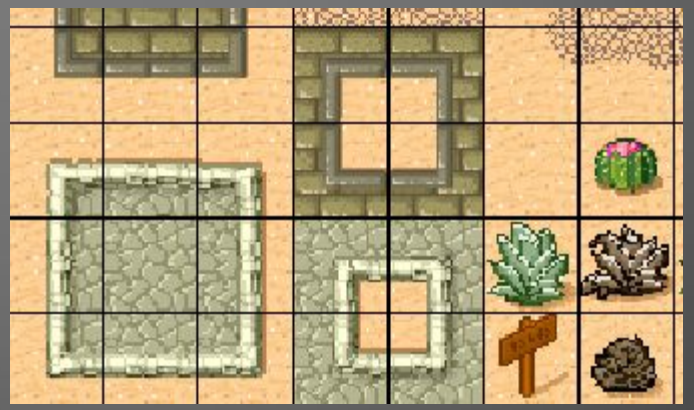
导入图块



- 新建图层

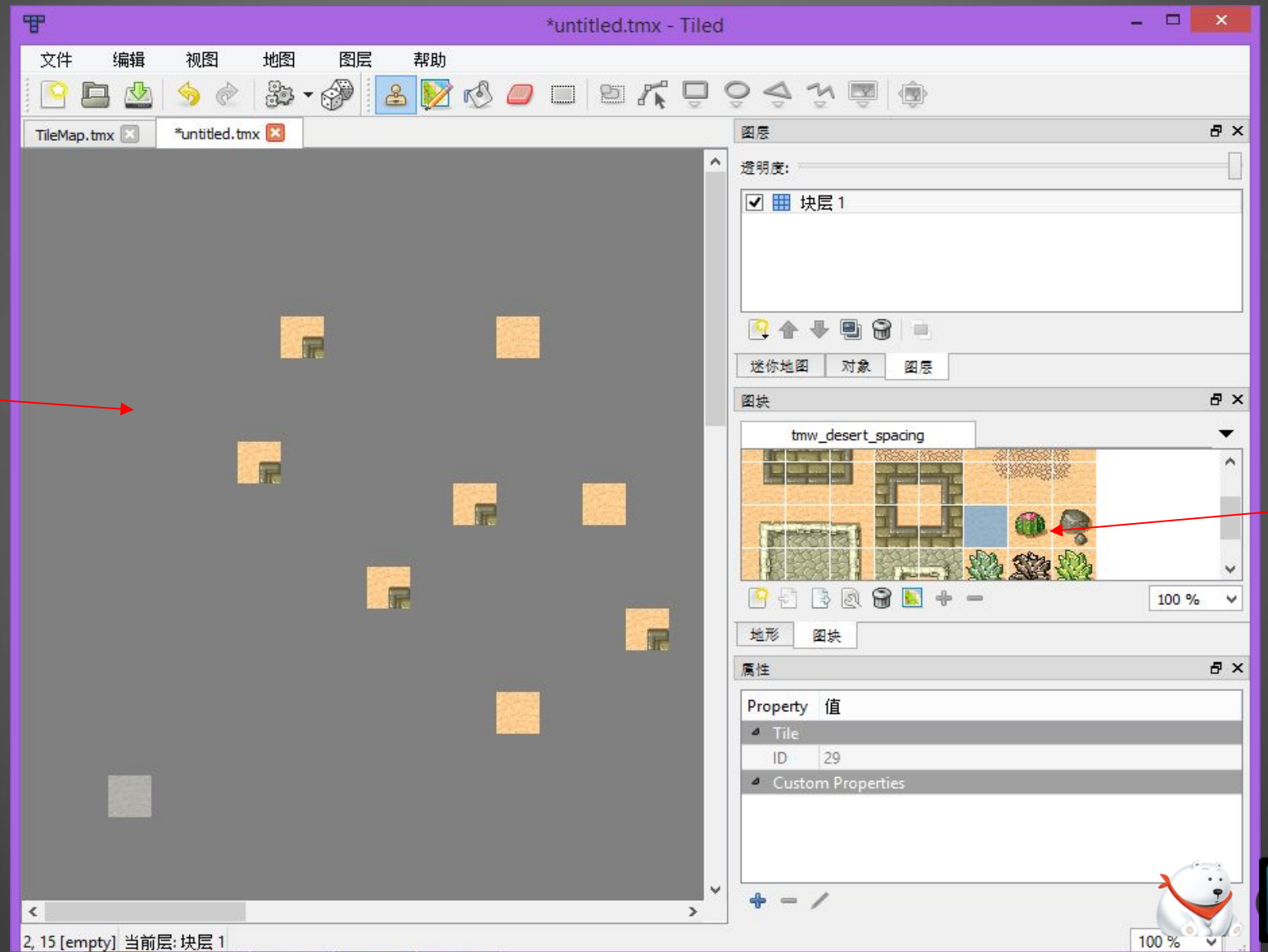
- 素材原因，这里边距，间距设为1个像素点

- 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("PushBox/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

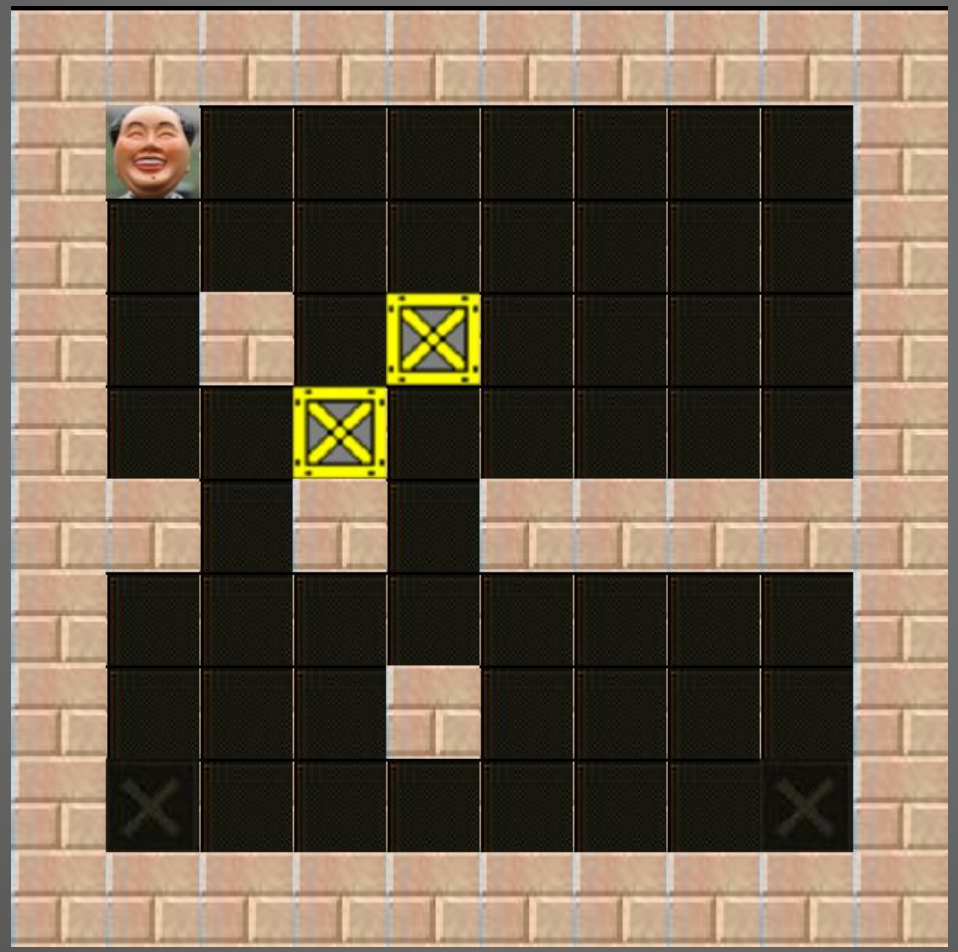
```
//从tmx中获取对象层
TMXObjectGroup* objects = map->getObjectGroup("box");

//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。

实现效果：



Tilemap使用课堂演示

hw: 做一个推箱子的小游戏

要求：

1：背景必须用Tilemap实现

2：用上Vectoc或Map其中一种数据结构存储管理对象

3：胜利所用最少步数记录以本地数据形式保存，每次运行游戏可以显示记录

游戏方式：

1：仿照传统推箱子游戏，把箱子推到指定地点即胜利。

2：用按钮W，S，A，D控制人物上下左右移动（亦可自行用其他任何方式实现）

3：鼓励玩法创新，可适当加分。

拓展知识---json

JSON---JavaScript Object Notation

- 一种轻量级的数据交换格式
- 理想的数据交换语言
- 以键值对保存数据

为什么要使用JSON？

JSON作为一种轻量级的数据交换格式，使用简单，易于人阅读和编写，同时也易于机器解析和生成，而且相对于XML格式更节省容量，特别适合用于信息传输，可以简便快捷地满足蓝牙传数据输，因特网数据传输等方面的要求。

JSON 语法规则

JSON 语法是 JavaScript 对象表示语法的子集。

- 数据在名称/值对中
- 数据由逗号分隔
- 花括号保存对象
- 方括号保存数组

```
{"json":"json string","array":[{"int":1,"double":1,"bool":true,"say":"hi"}]}
```

COCOS2DX

THE END

THANKS FOR WATCHING