

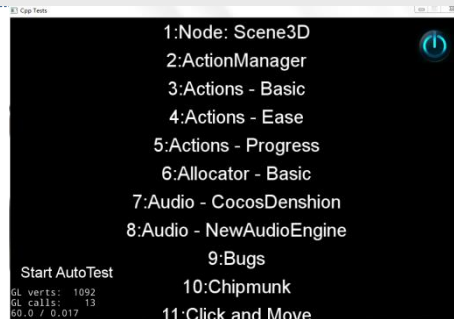
第五章 二维游戏场景绘制

上节回顾

- Cocos2d-x基本概念
 - 组件构成、导演、场景、精灵
- Cocos2d-x坐标系
 - 各种坐标系、锚点
- Cocos2d-x基础类
 - 导演、节点、场景、层、精灵、菜单
- Cocos2d-x开发环境配置
 - Windows下开发环境配置

本节目录

- 两个实例，一张地图
 - 官方测试项目 `cpp-tests`



- 永远的 HelloWorld

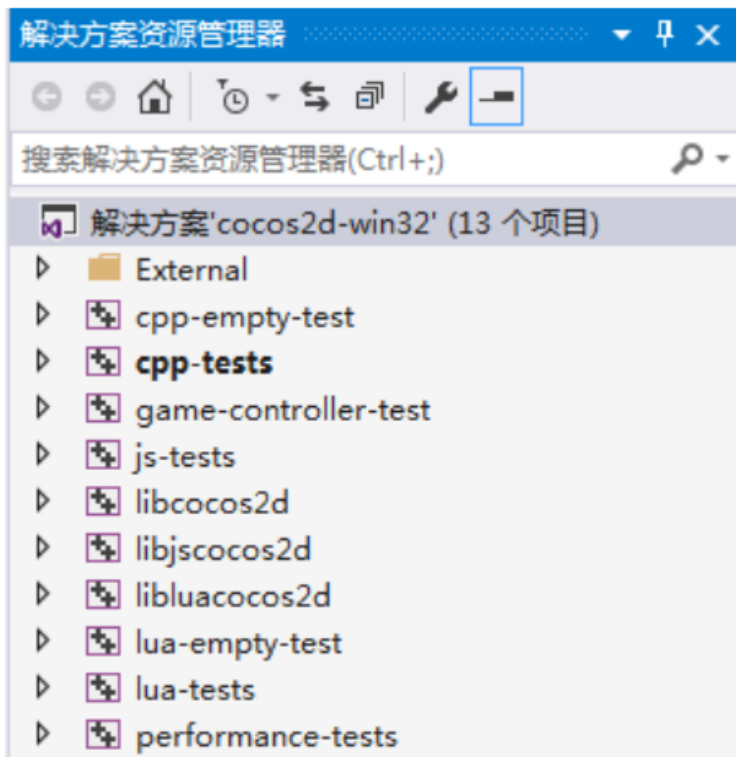


- 瓦片地图



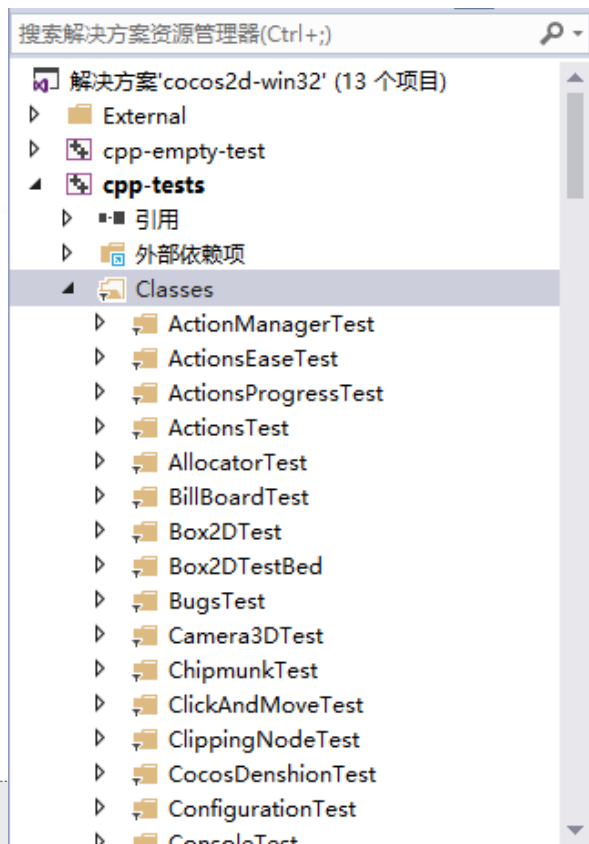
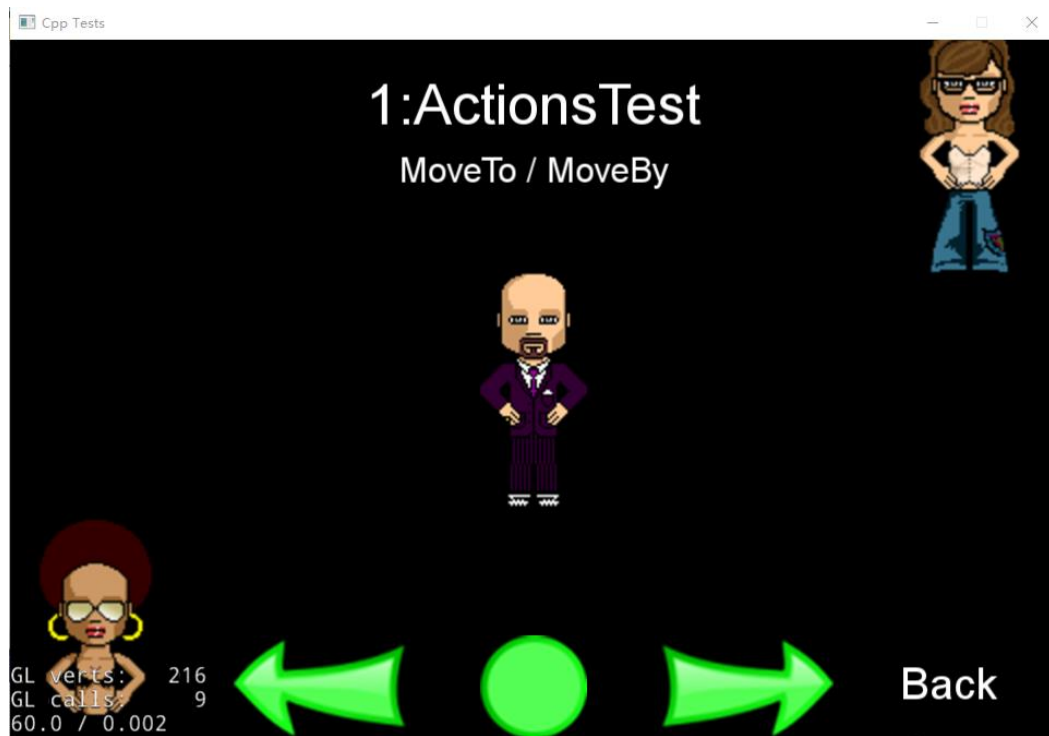
官方测试项目 cpp-tests

双击 `cocos2d-x-3.17\build\cocos2d-win32.sln`，Visual Studio 将打开此解决方案，解决方案打开后，可以看到这样的项目列表：



官方测试项目 cpp-tests

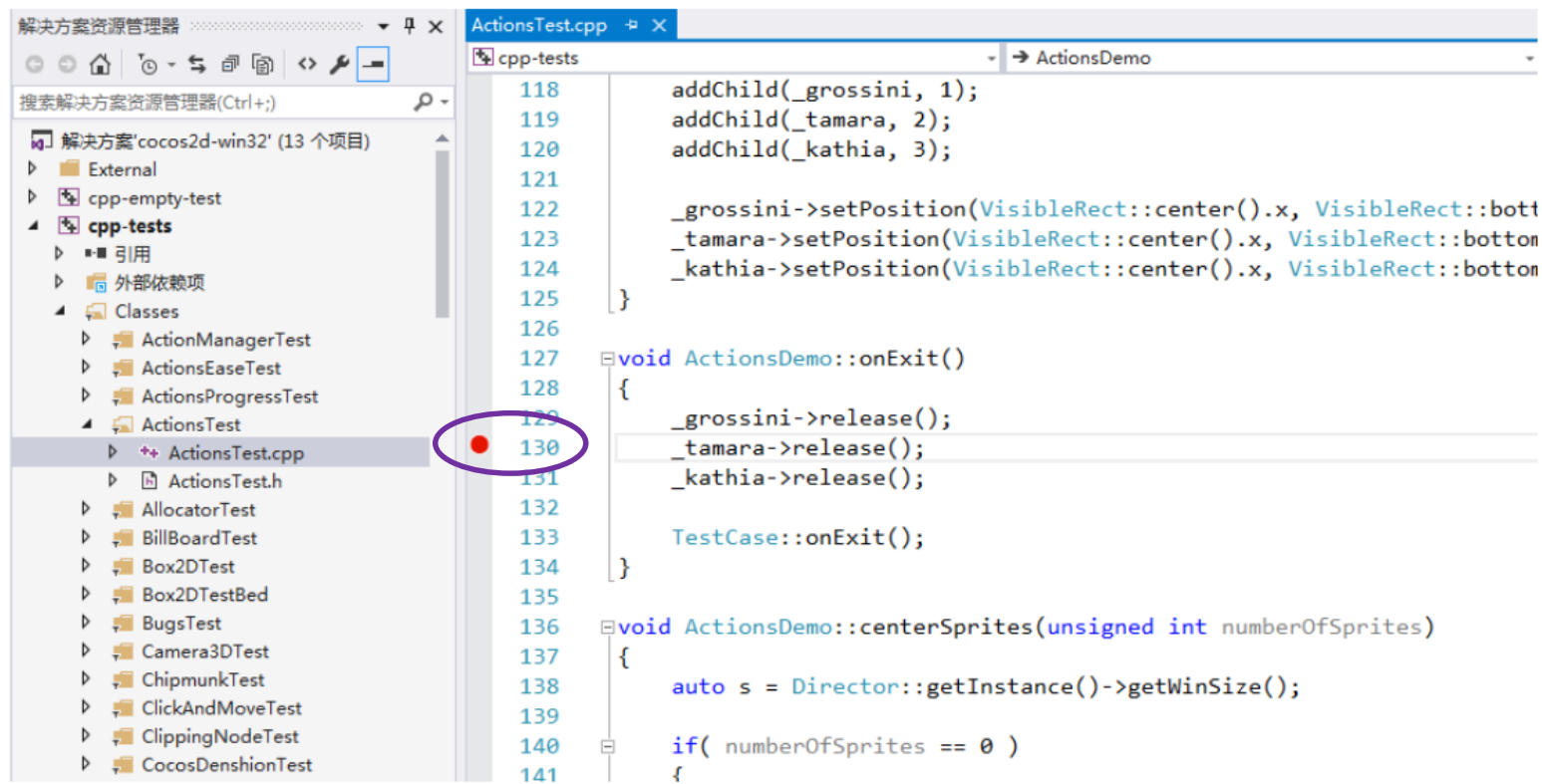
默认情况下项目列表中 `cpp-tests` 加粗显示，表示是启动项目，此时点击菜单栏中 本地 Windows 调试器 进行项目的编译和运行。编译过程视机器性能不同，会花费 10-30 分钟的时间，编译完成后，将自动运行，运行成功将看到测试程序：



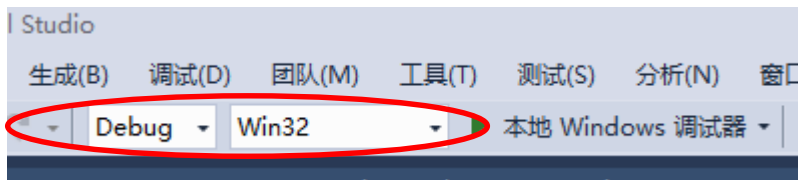
官方测试项目 cpp-tests

如何调试(Debug)

1. 点击代码行左侧的空白，设置断点



官方测试项目 cpp-tests



2. 以 debug 模式运行 cpp-tests

3. 操作 App 触发断点，IDE 将卡在断点处，Debug 视图会自动跳出，可以查看运行堆栈和变量的值

The screenshot shows the Visual Studio IDE with the '自动窗口' (Automatic Windows) and '调用堆栈' (Call Stack) views open. The '自动窗口' view shows the current state of the program, including the 'this' pointer and the 'TestCase' object. The '调用堆栈' view shows the call stack, including the entry point 'cpp-tests.exe!ActionsDemo::onExit()' and the 'libcocos2d.dll' library.

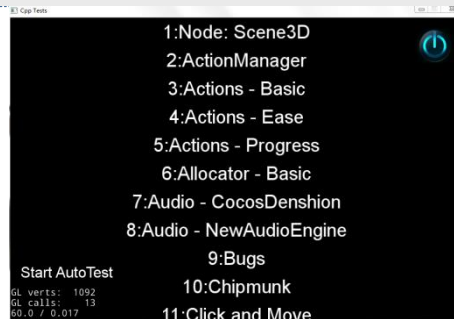
名称	值	类型
_grossini	0x0d337458 [_textureAtlas=0x00000000 <NULL> _atlasIndex=-1 _t	cocos2d
_tamara	0x0d337920 [_textureAtlas=0x00000000 <NULL> _atlasIndex=-1 _t	cocos2d
this	0x05a0e498 {...}	ActionsC
[ActionMove]	{...}	ActionM
TestCase	{_priorTestItem=0x0d2965c8 {...} _restartTestItem=0x05a12600 {...} Test	TestCase
cocos2d::Scene	{_cameras={ size=1 } _defaultCamera=0x05a10f08 [_scene=0x05a1	cocos2d
_priorTestItem	0x0d2965c8 {...}	cocos2d
_restartTestItem	0x05a12600 {...}	cocos2d
_nextTestItem	0x0d33d2c8 {...}	cocos2d
_titleLabel	0x05a11cd8 {_currentLabelType=TTF (0) _contentDirty=false _utf:	cocos2d
_subtitleLabel	0x0d2238b8 {_currentLabelType=TTF (0) _contentDirty=false _utf:	cocos2d
_testSuite	0x0d155088 {...}	TestSuite
_runTime	1.18846333	float
_testCaseName	"ActionMove"	std::basi

名称
cpp-tests.exe!ActionsDemo::onExit() 行 130
libcocos2d.dll!cocos2d::Director::setNextScene() 行 1194
libcocos2d.dll!cocos2d::Director::drawScene() 行 315
libcocos2d.dll!cocos2d::Director::mainLoop() 行 1501
libcocos2d.dll!cocos2d::Application::run() 行 112
cpp-tests.exe!wWinMain(HINSTANCE__ * hInstance, HINSTANCE__ * hPrevInstance, wchar_t *
[外部代码]
[下面的框架可能不正确和/或缺失，没有为 kernel32.dll 加载符号]

转场前

本节目录

- 两个实例，一张地图
 - 官方测试项目 cpp-tests



- 永远的HelloWorld



- 瓦片地图



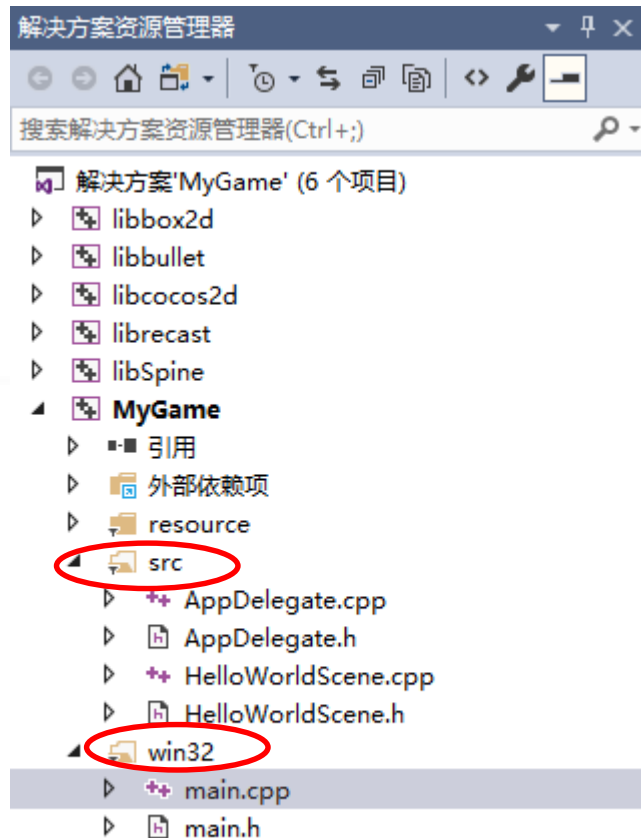
永远的HelloWorld

- main 对应平台的代码

- AppDelegate

- HelloWorldScene

我们自己的实现代码



永远的HelloWorld

- main.cpp
- #define USING_NS_CC
- using namespace cocos2d
- 使用cocos2d-x的命名空间



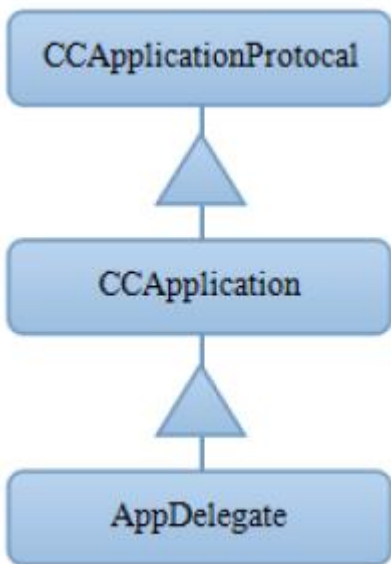
HelloWorld : public cocos2d::Scene

if (!Scene::init())

```
AppDelegate.cpp  AppDelegate.h  main.h  main.cpp  HelloWorldScene.cpp  Hello
ame (全局范围)
1  #include "main.h"
2  #include "AppDelegate.h"
3  #include "cocos2d.h"
4
5  USING_NS_CC;
6
7  int WINAPI _tWinMain(HINSTANCE hInstance,
8                      HINSTANCE hPrevInstance,
9                      LPTSTR lpCmdLine,
10                     int nCmdShow)
11  {
12      UNREFERENCED_PARAMETER(hPrevInstance);
13      UNREFERENCED_PARAMETER(lpCmdLine);
14
15      // create the application instance
16      AppDelegate app;
17      return Application::getInstance()->run();
18  }
19
```

永远的HelloWorld

- main.cpp
- 定义Windows程序入口函数



```
AppDelegate.cpp  AppDelegate.h  main.h  main.cpp  HelloWorldScene.cpp  Hello
ame (全局范围)
1  #include "main.h"
2  #include "AppDelegate.h"
3  #include "cocos2d.h"
4
5  USING_NS_CC;
6
7  int WINAPI _tWinMain(HINSTANCE hInstance,
8                      HINSTANCE hPrevInstance,
9                      LPTSTR lpCmdLine,
10                     int nCmdShow)
11  {
12      UNREFERENCED_PARAMETER(hPrevInstance);
13      UNREFERENCED_PARAMETER(lpCmdLine);
14
15      // create the application instance
16      AppDelegate app;
17      return Application::getInstance()->run();
18  }
19
```

单例模式!

显示窗口，进入消息循环

永远的HelloWorld

• AppDelegate.h

```
AppDelegate.h  AppDelegate.h  main.h  main.cpp  HelloWorldScene.cpp
AppDelegate.h  (全局范围)
19  /**
20  @brief  Implement Director and Scene init code here.
21  @return true  Initialize success, app continue.
22  @return false  Initialize failed, app terminate.
23  */
24  virtual bool applicationDidFinishLaunching();
25
26  /**
27  @brief  Called when the application moves to the background
28  @param  the pointer of the application
29  */
30  virtual void applicationDidEnterBackground();
31
32  /**
33  @brief  Called when the application reenters the foreground
34  @param  the pointer of the application
35  */
36  virtual void applicationWillEnterForeground();
37  };
38
39  #endif // _APP_DELEGATE_H_
```

```
int Application::run()
{
    if (applicationDidFinishLaunching())
    {
        [[CCDirectorCaller sharedDirectorCaller] startMainLoop];
    }
    return 0;
}
```

因此，这个方法执行完成后，Cocos2d-x就开始了主逻辑循环。

**执行Application ::
run 函数时被调用**



永远的HelloWorld

- AppDelegate.cpp
 - applicationDidFinishLaunching()

1

- 初始化导演

```
56  bool AppDelegate::applicationDidFinishLaunching() {  
57      // initialize director  
58      auto director = Director::getInstance();
```

2

- 创建第一个场景

```
96      // create a scene. it's an autorelease object  
97      auto scene = HelloWorld::createScene();
```

3

- 启动这个场景

```
99      // run  
100     director->runWithScene(scene);
```

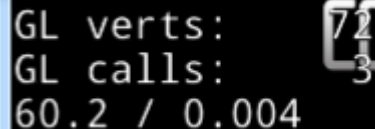
永远的HelloWorld

applicationDidFinishLaunching()

- 导演的工作

```
59     auto glview = director->getOpenGLView();
60     if(!glview) {
61     #if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN32) || (CC_TARGET_PLATFORM == C
62         glview = GLViewImpl::createWithRect("MyGame", cocos2d::Rect(0, 0,
63     #else
64         glview = GLViewImpl::create("MyGame");
65     #endif
66     director->setOpenGLView(glview);
67     }
```

```
69     // turn on display FPS
70     director->setDisplayStats(true);
72     // set FPS. the default value is 1.0/60 if you don't call this
73     director->setAnimationInterval(1.0f / 60);
75     // Set the design resolution
76     glview->setDesignResolutionSize(designResolutionSize.width,
```

A screenshot of a game's FPS display overlay. It shows 'GL verts: 72' and 'GL calls: 3' in a large font, with '60.2 / 0.004' in a smaller font below. The background is black with white text. The 'COC' logo is partially visible on the right.

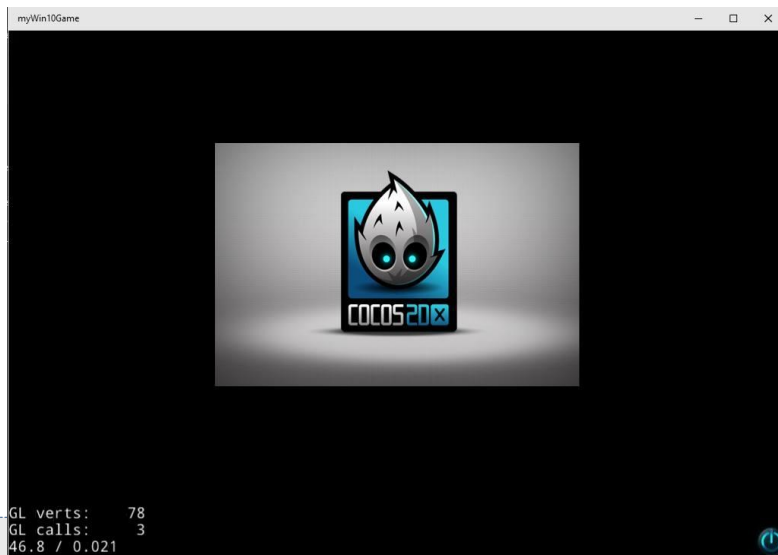
GL verts:	72
GL calls:	3
60.2 / 0.004	

永远的HelloWorld

applicationDidFinishLaunching()

- 分辨率

```
21 static cocos2d::Size designResolutionSize = cocos2d::Size(480, 320);
22 static cocos2d::Size smallResolutionSize = cocos2d::Size(480, 320);
23 static cocos2d::Size mediumResolutionSize = cocos2d::Size(1024, 768);
24 static cocos2d::Size largeResolutionSize = cocos2d::Size(2048, 1536);
```



永远的HelloWorld

- HelloWorldScene.h

```
ate.cpp  AppDelegate.h  main.h  main.cpp  HelloWorldScene.cpp  HelloWorldScene.h  X
ne
1  #ifndef __HELLOWORLD_SCENE_H__
2      #define __HELLOWORLD_SCENE_H__
3
4      #include "cocos2d.h"
5
6      class HelloWorld : public cocos2d::Scene
7      {
8      public:
9          static cocos2d::Scene* createScene(); 1
10
11          virtual bool init(); 2
12
13          // a selector callback
14          void menuCloseCallback(cocos2d::Ref* pSender); 3
15
16          // implement the "static create()" method manually
17          CREATE_FUNC(HelloWorld); 4
18      };
19
20      #endif // __HELLOWORLD_SCENE_H__
```

永远的HelloWorld

- HelloWorldScene.cpp

– applicationDidFinishLaunching()

2

- 创建第一个场景

```
96 // create a scene. it's an autorelease object
97 auto scene = HelloWorld::createScene();
```

```
6  Scene* HelloWorld::createScene()
7  {
8      return HelloWorld::create();
9  }
```



- HelloWorldScene.h

```
16 // implement the "static create()" method manually
17 4 CREATE_FUNC(HelloWorld);
```

永远的HelloWorld

• CREATE_FUNC

```
1 #define CREATE_FUNC(__TYPE__) \
2 static __TYPE__ create() \
3 { \
4     __TYPE__ *pRet = new __TYPE__(); \
5     if (pRet && pRet->init()) \
6     { \
7         pRet->autorelease(); \
8         return pRet; \
9     } \
10    else \
11    { \
12        delete pRet; \
13        pRet = NULL; \
14        return NULL; \
15    } \
16 }
```

```
17 static HelloWorld* create()
18 {
19     HelloWorld *pRet = new HelloWorld();
20     if (pRet && pRet->init())
21     {
22         pRet->autorelease();
23         return pRet;
24     }
25     else
26     {
27         delete pRet;
28         pRet = NULL;
29         return NULL;
30     }
31 }
32 ;;
```

永远的HelloWorld

HelloWorldScene.cpp

- init()

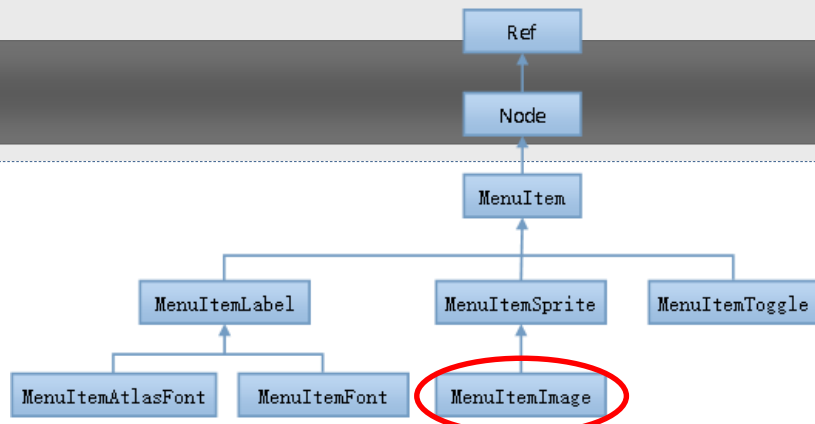
```
11 // on "init" you need to initialize your instance
12 bool HelloWorld::init()
13 {
14     //////////////////////////////////////
15     // 1. super init first
16     if ( !Scene::init() )
17     {
18         return false;
19     }
20
21     auto visibleSize = Director::getInstance()->getVisibleSize();
22     Vec2 origin = Director::getInstance()->getVisibleOrigin();
```



永远的HelloWorld

HelloWorldScene.cpp

- init()



```
25 // 2. add a menu item with "X" image, which is clicked to quit the program
26 //    you may modify it.
```

```
27
28 // add a "close" icon to exit the progress. it's an autorelease object
```

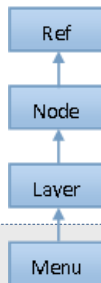
```
29 auto closeItem = MenuItemImage::create(
30     "CloseNormal.png",
31     "CloseSelected.png",
32     CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));
33
```

```
Director::getInstance()->end();
```

```
34 closeItem->setPosition(Vec2(origin.x + visibleSize.width - closeItem->getContentSize().width/2 ,
35     origin.y + closeItem->getContentSize().height/2));
```

```
36
37 // create menu, it's an autorelease object
```

```
38 auto menu = Menu::create(closeItem, NULL);
39 menu->setPosition(Vec2::ZERO);
40 this->addChild(menu, 1);
```



永远的HelloWorld

HelloWorldScene.cpp

- init()

```
42 //////////////////////////////////////////////////
43 // 3. add your codes below...
44
45 // add a label shows "Hello World"
46 // create and initialize a label
47
48 //auto label = Label::createWithTTF("Hello World", "fonts/Marker Felt.ttf", 24);
49 auto label = Label::createWithTTF("My Game", "fonts/Marker Felt.ttf", 36);
50
51 // position the label on the center of the screen
52 //label->setPosition(Vec2(origin.x + visibleSize.width/2,
53 //                        //origin.y + visibleSize.height - label->getContentSize().height));
54 label->setPosition(Vec2(origin.x + visibleSize.width / 2,
55                          origin.y + visibleSize.height / 2));
56
57
58 // add the label as a child to this layer
59 this->addChild(label, 1);
```

MyGame > Resources > fonts

名称

arial.ttf

Marker Felt.ttf

永远的HelloWorld

- 标签 (Label)
 - 位图字体 (BMFont)
 - TrueType 字体 (TTF)
 - 系统字体 (SystemFont)



永远的HelloWorld

– 位图字体 (BMFont)

BMFont 是一个使用位图字体创建的标签类型，位图字体中的字符由点阵组成。使用这种字体标签性能非常好，但是不适合缩放。由于点阵的原因，缩放会导致失真。标签中的每一个字符都是一个单独的 Sprite，也就是说精灵的属性(旋转，缩放，着色等)控制都适用于这里的每个字符。

```
auto myLabel = Label::createWithBMFont("bitmapRed.fnt", "Your Text");
```

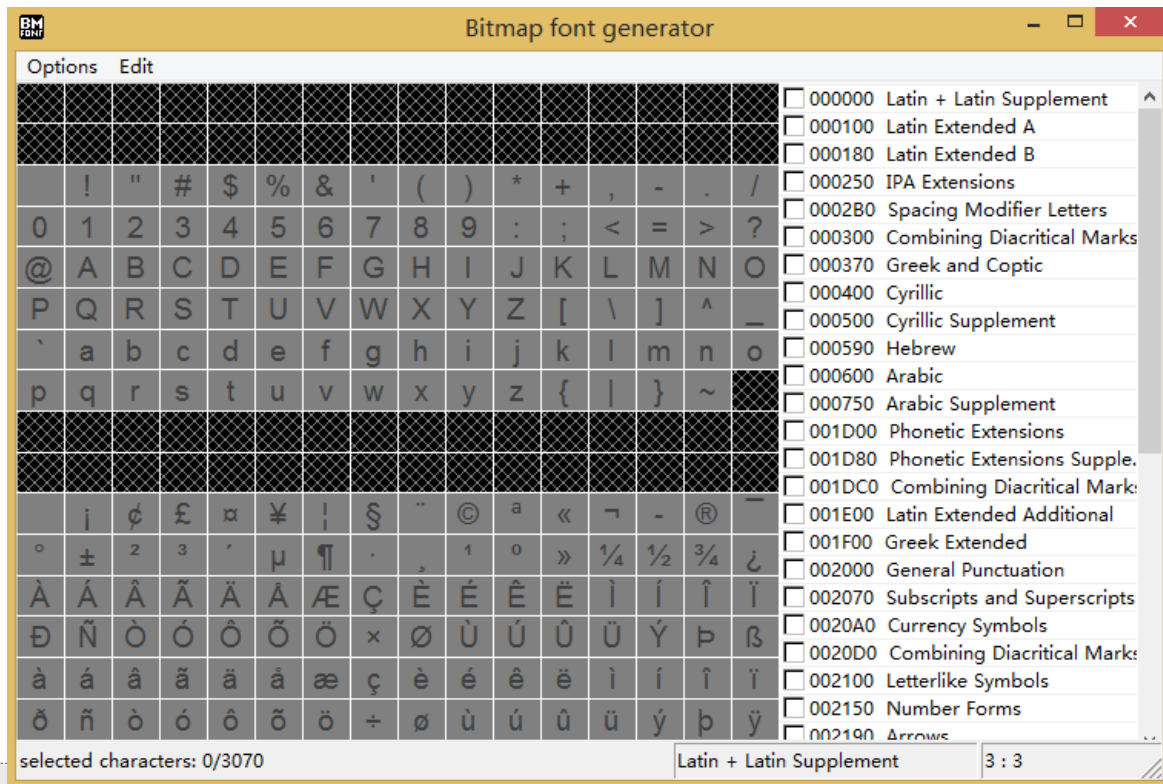


LabelBMFont
LabelTTF
LabelTTF from TTFConfig
Label using SystemFont
LabelTTF with Shadow
LabelTTF with Outline
LabelTTF with Glow

永远的HelloWorld

– 位图字体 (BMFont)

创建 BMFont 标签需要两个文件: .fnt 文件和 .png 文件



Latin.fnt
Latin_0.png



永远的HelloWorld

转场前

永远的HelloWorld

– TrueType 字体 (TTF)

TrueType 字体和我们上面了解的位图字体不同，使用这种字体很方便，你不需要为每种尺寸和颜色单独使用字体文件。不像 BMFont，如果想不失真的缩放，就要提供多种字体文件。

要创建这种标签，需要指定 **.ttf** 字体文件名，文本字符串和字体大小。

```
auto myLabel = Label::createWithTTF("Your Text", "Marker Felt.ttf", 24);
```



LabelBMFont
LabelTTF
LabelTTF from TTFConfig
Label using SystemFont
LabelTTF with Shadow
LabelTTF with Outline
LabelTTF with Glow

永远的HelloWorld

– TrueType 字体 (TTF)

虽然使用 TrueType 字体比使用位图字体更灵活, 但是它渲染速度较慢, 并且更改标签的属性(字体, 大小)是一项非常消耗性能的操作。

如果您需要具有相同属性的多个 Label 对象, 那可以创建一个 `TTFConfig` 对象来统一配置, `TTFConfig` 对象允许你设置所有标签的共同属性。

```
// create a TTFConfig files for labels to share
```

```
TTFConfig labelConfig;
```

```
labelConfig.fontFilePath = "myFont.ttf";
```

```
labelConfig.fontSize = 16;
```

```
labelConfig.glyphs = GlyphCollection::DYNAMIC;
```

```
labelConfig.outlineSize = 0;
```

```
labelConfig.customGlyphs = nullptr;
```

```
labelConfig.distanceFieldEnabled = false;
```

```
// create a TTF Label from the TTFConfig file.
```

```
auto myLabel = Label::createWithTTF(labelConfig, "My Label Text");
```

LabelBMFont
LabelTTF
LabelTTF from TTFConfig ←
Label using SystemFont
LabelTTF with Shadow
LabelTTF with Outline
LabelTTF with Glow

永远的HelloWorld

– 系统字体 (SystemFont)

`SystemFont` 是一个使用系统默认字体，默认字体大小的标签类型，这样的标签不要改变他的属性，它会使用系统的规则。

```
auto myLabel = Label::createWithSystemFont("My Label Text", "Arial", 16);
```

LabelBMFont
LabelTTF
LabelTTF from TTFConfig
Label using SystemFont
LabelTTF with Shadow
LabelTTF with Outline
LabelTTF with Glow



永远的HelloWorld

- 标签效果

在屏幕上有标签后，它们可能看起来很普通，这时你希望让他们变漂亮。你不用创建自定义字体! Label 对象就可以对标签应用效果，包括阴影，描边，发光。

```
// shadow effect is supported by all Label types  
myLabel->enableShadow();
```



LabelBMFont
LabelTTF
LabelTTF from TTFConfig
Label using SystemFont
LabelTTF with Shadow
LabelTTF with Outline
LabelTTF with Glow

永远的HelloWorld

- 标签效果

```
// outline effect is TTF only, specify the outline color desired  
myLabel->enableOutline(Color4B::WHITE, 1));
```

```
// glow effect is TTF only, specify the glow color desired.  
myLabel->enableGlow(Color4B::YELLOW);
```

LabelBMFont
LabelTTF
LabelTTF from TTFConfig
Label using SystemFont
LabelTTF with Shadow
LabelTTF with Outline
LabelTTF with Glow



永远的HelloWorld

HelloWorldScene

- init()

```
61 // add "HelloWorld" splash screen"
62 //auto sprite = Sprite::create("HelloWorld.png");
63 auto sprite = Sprite::create("mysprite.png");
64
65 // position the sprite on the center of the screen
66 //sprite->setPosition(Vec2(visibleSize.width/2 + origin.x, visibleSize.height/2 + origin.y));
67 //sprite->setPosition(Vec2(100, 100));
68 auto w = sprite->getContentSize().width;
69 auto h = sprite->getContentSize().height;
70 sprite->setPosition(Vec2(w / 2 + origin.x, h / 2 + origin.y));
71
72 // add the sprite as a child to this layer
73 this->addChild(sprite, 0);
```

Resources



fonts



res



CloseNormal.png



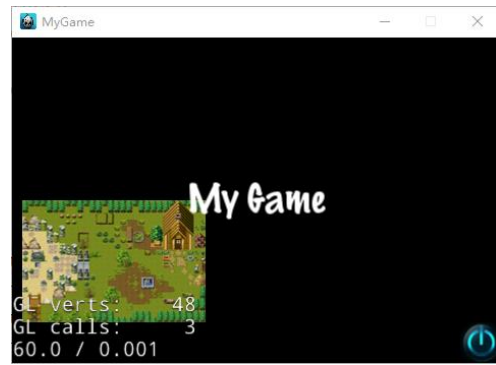
CloseSelected.png



HelloWorld.png

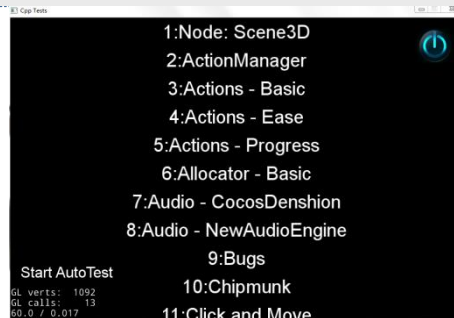


mysprite.png



本节目录

- 两个实例，一张地图
 - 官方测试项目 cpp-tests



- 永远的 HelloWorld



- 瓦片地图



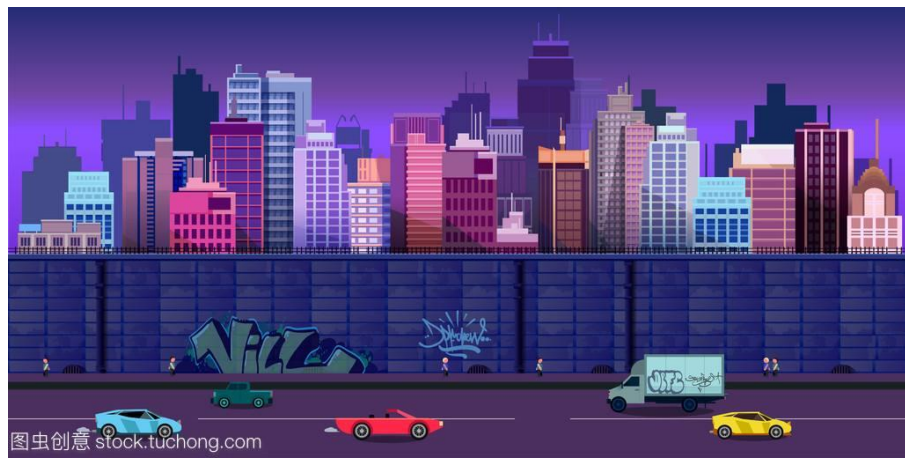
游戏画面背景

- 游戏画面特点

- 画面质量直接影响游戏销量
- 不是单一的图片组成
- 分层次显示
 - 前景、远景、底色和建筑层等

- 二维游戏背景类型

- 平面滚动型
- 斜45度型
- 俯视角型



二维游戏背景类型

- 平面滚动型

- 大多数二维游戏产品采用的类型

- 多用于冒险类和设计类的游戏

- 特点

- 保持游戏主角不动
 - 游戏背景向反方向运动
 - 使游戏主角与背景产生相对运动
 - 从而制作出动画效果



二维游戏背景类型

- 斜45度型
 - 主要用于策略类和角色扮演类游戏
 - 允许开发者创建壮观的、详细的视觉效果
 - 常见于一些日韩的角色扮演类游戏中
- 特点
 - 背景中的物体具有层次感
 - 有视觉上的遮挡关系



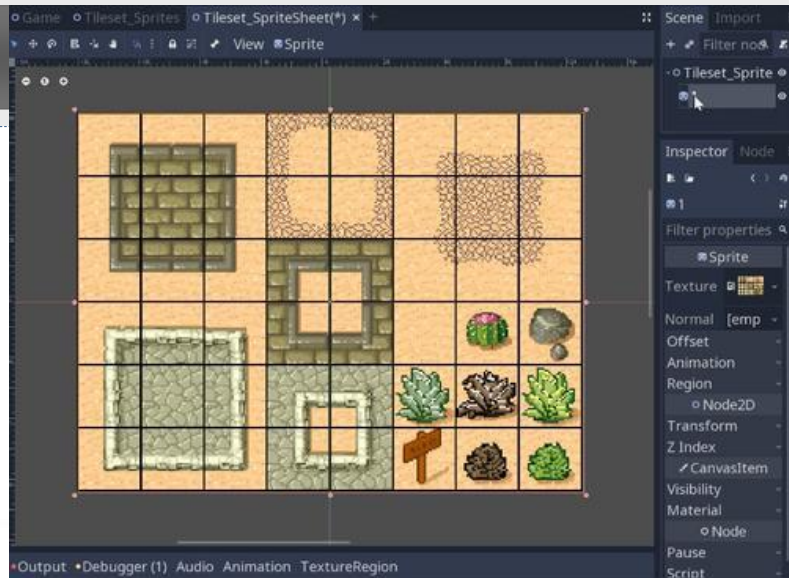
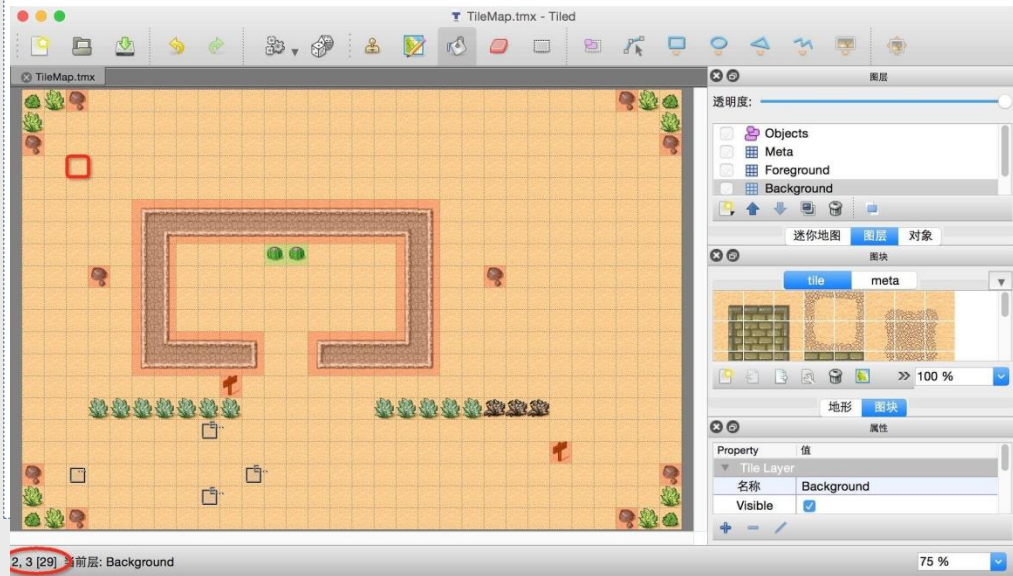
二维游戏背景类型

- 俯视角型
 - 常见于棋牌类和策略类的游戏
 - 特点
 - 站在俯视的角度观察整个游戏



游戏地图的创建

- 固定地图
 - 使用**固定**背景
 - 屏幕切割成**棋盘**状
 - 内存中保持**小块编号**
 - 小块**拼接**成背景地图



1	3	3	4	1	1	2
1	0	0	2	2	0	1
1	0	0	2	2	0	0
1	0	0	0	3	0	3
1	0	0	2	2	0	2
0	0	0	0	0	0	2
3	3	3	3	1	2	2

地图数据

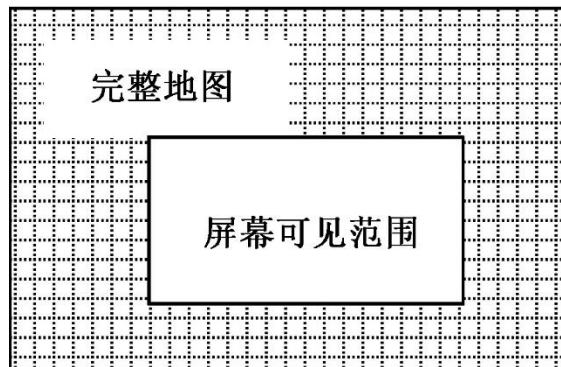
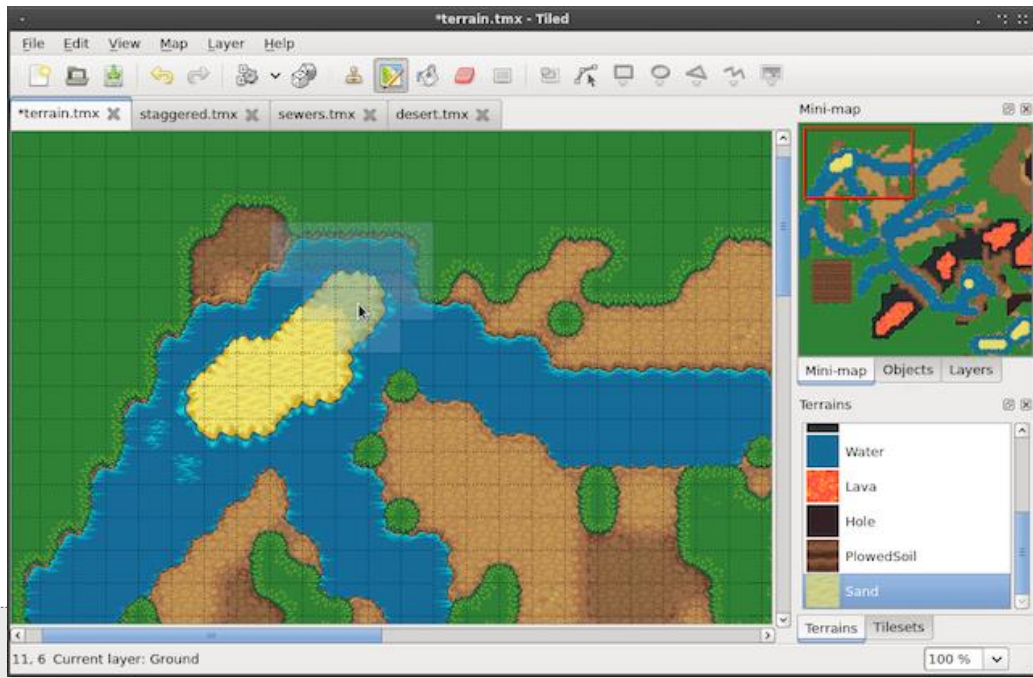
图片 (及其编号)

0	1	2	3	4
---	---	---	---	---	-------

游戏地图的创建

- 滚屏地图

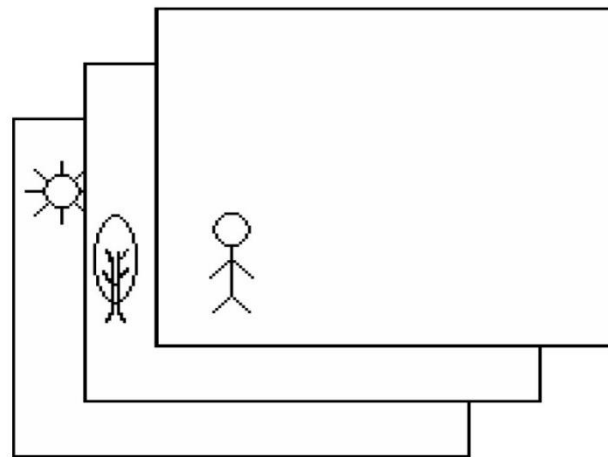
- 是固定地图的扩展
- 可以显示远大于地图的图像
- 根据角色当前位置显示图像中对应区域地图



游戏地图的创建

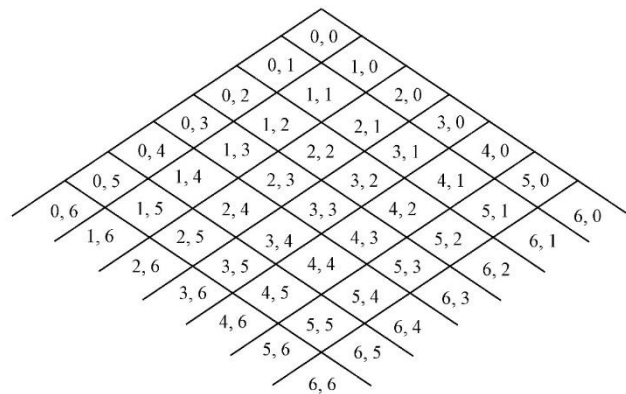
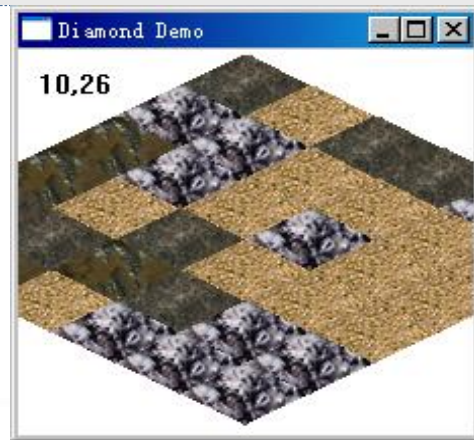
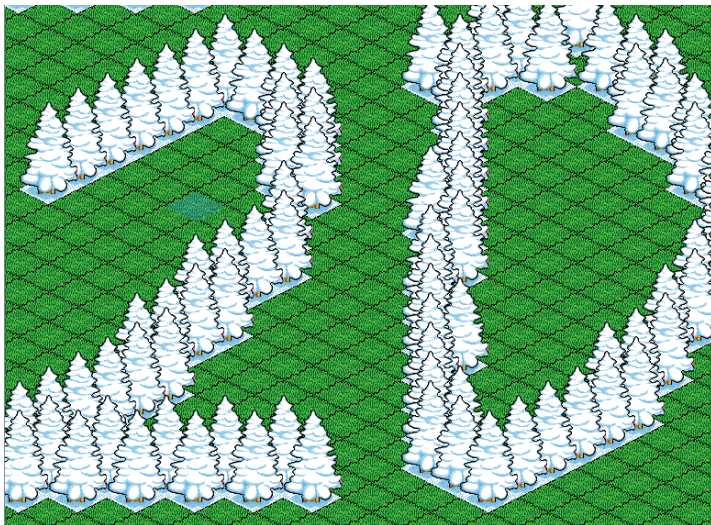
• 多层地图

- 小地图能够重叠或者有层次关系
- 在背景上有多个物体运动
- 可以模拟物体远近不同的透视关系



游戏地图的创建

- 菱形地图
 - 使用二维模拟三维效果
 - 使用小的菱形地图块拼接形成



游戏地图的创建

- Cocos2d-x中的地图
 - 一张大的背景图可以由几种地形来表示
 - 每种地形对应一张小的图片
 - 这些小的地形图片称之为瓦片
 - 把这些瓦片拼接在一起，一个完整的地图就组合出来了
- 下载瓦片地图编辑器
 - <https://www.mapeditor.org/>



游戏地图的创建

- 瓦片地图编辑器 – Tiled Map Editor

- 步骤1, 新建地图:

- ✓ “文件” → “新建” → “创建新地图”

- ✓ 创建时可设置

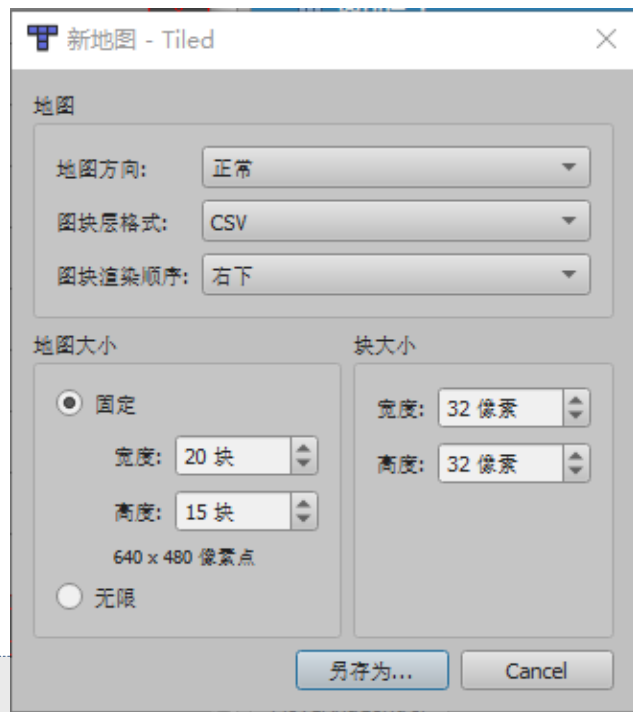
- ✓ 地图方向 (块排列方向)

- ✓ 地图大小 (宽高的块数)

- ✓ 块大小

- ✓ 保存为Tiled地图文件

- ✓ Tmx格式



游戏地图的创建

• 瓦片地图编辑器 – Tiled Map Editor

– 步骤2，新建图块：

✓ “新建” → “新图块”

✓ 在弹出框中可设置

✓ 图块名称

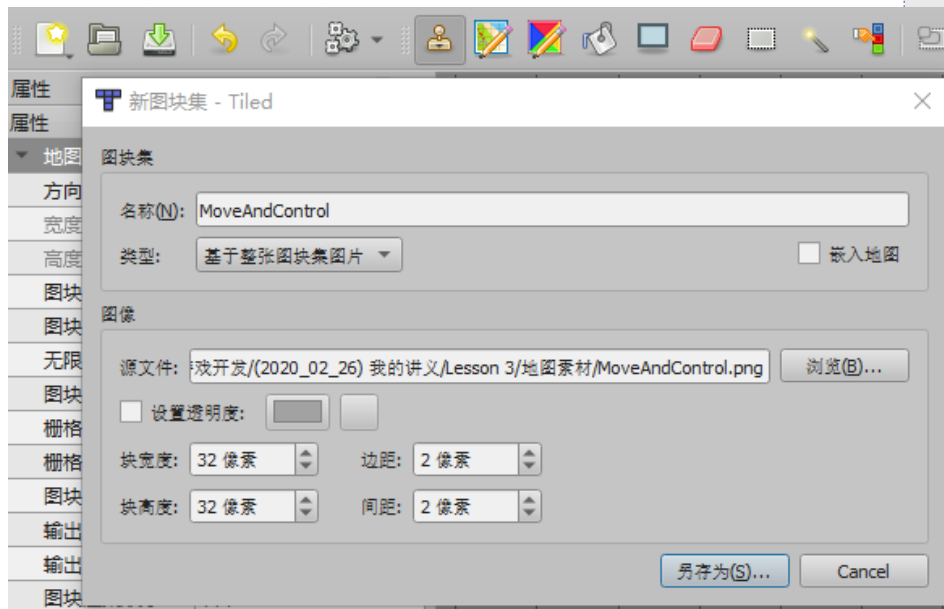
✓ 素材源

✓ 素材切割块大小

✓ 块间距

✓ 保存为Tiled图块集文件

✓ Tsx格式



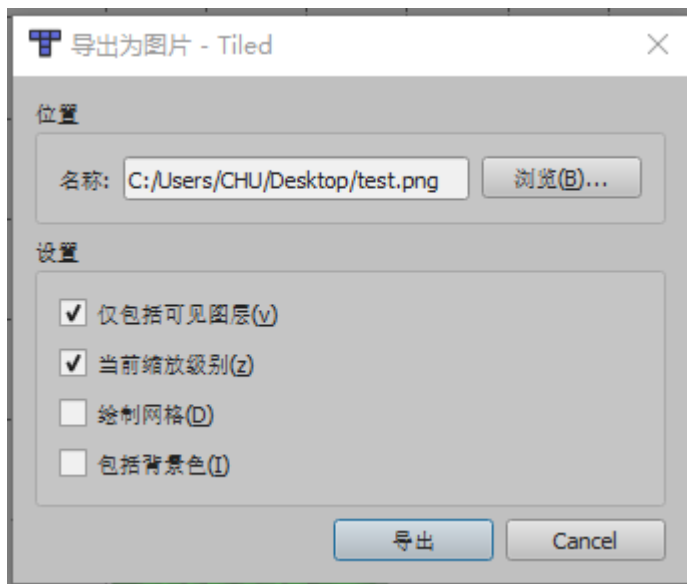
游戏地图的创建

- 瓦片地图编辑器 – Tiled Map Editor
 - 步骤3，绘制地图：



游戏地图的创建

- 瓦片地图编辑器 – Tiled Map Editor
 - 步骤4，导出并保存地图：
 - ✓ “文件” → “新建” → “导出为图片”

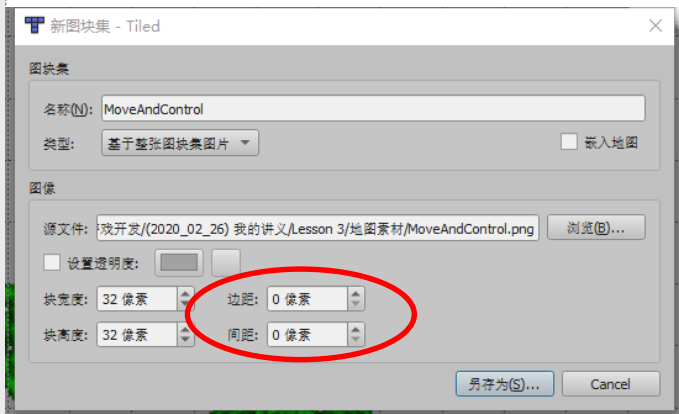


游戏地图的创建

- 瓦片地图编辑器 – Tiled Map Editor

- 易错点

- 块间距



游戏地图的创建

– 本周作业：

- ✓ 下载Tile Map Editor并安装
- ✓ 利用提供的“地图素材”（或自行搜索、下载）绘制两幅二维游戏背景地图
- ✓ 主题分别为“室外”和“室内”
- ✓ 用生成的PNG图片替换“Hello World”项目中图片，分别显示在窗口的左下角和右上角
 - ✓ 提示：如果图片太大，窗口无法完整显示，应设法修改窗口大小或图片分辨率
- ✓ 在窗口左上角和右下角分别显示自己的学号和拼音名
- ✓ 将两幅地图图片和运行后的窗口截图上传至BB系统