1 选题

关于练手项目，选择时考虑的因素：

1. 主要目的是将所需 C++ 知识、开发过程、软件系统分析与设计能力融汇贯通。达成，植物大战僵尸项目用到了上述知识。
2. 项目的目标不要高，开发该项目不要涉及过多的自己不懂的专业知识。达成，植物大战僵尸项目使用 cocos2dx，学习成本较低。
3. 项目的规模不要大，代码行在 1000~2000 行，开发周期在半个月左右。未达成，植物大战僵尸项目的规模远远超乎想象。

参考以上三点因素，得出项目规模等同于期末大作业的结论，因此将搜索范围划定为各高校 C++ 期末大作业。另外考虑到个人兴趣，决定进行游戏开发，在这个方向上有同济大学和中国人民大学的大作业文档作为参考，大作业文档存放在 Game/docs。

https://www.zhihu.com/question/393933369/answer/1216606194，同济大学 C++ 大作业，开心消消乐或元气骑士（团队开发）。

https://www.zhihu.com/question/391116845/answer/1207072945，中国人民大学 C++ 大作业，塔防游戏（单人开发）。

游戏感兴趣的选题有僵尸危机、植物大战僵尸和王国保卫战。其中对植物大战僵尸最为熟悉，并且该游戏有许多爱好者复现，资源丰富，因此决定开发本游戏。

1.1 僵尸危机（Boxhead）

参考同济大学大作业中的元气骑士，相似的游戏有曾经比较火的僵尸危机小游戏。该游戏采用 Flash 开发，已经过时，并且不再维护。由于该游戏画面简洁、规则简单，有爱好者自行开发了相似的游戏。

http://www.boxheadx.com/，Boxhead 官方网站，运行游戏需要 Flash。

https://github.com/emilegascoin/BoxHead，Java 开发的 Boxhead，有少量文档，但是界面过于简单，和实际游戏相差较大。

https://github.com/sdamghan/BoxHead，C++ 开发的 Boxhead 游戏，使用 SDL 库，代码行数在 3000-5000 之间，界面资源引自游戏原版素材，但是没有文档，参考价值不大。

https://github.com/EverardoG/boxhead_cpp，未完成开发的 Boxhead，目录结构比较清晰。

1.2 植物大战僵尸（Plants Vs Zombies）

参考中国人大大作业的塔防游戏，是个热门选题。植物大战僵尸可以说是人尽皆知的游戏，有广泛的用户群体和爱好者。该游戏已经有业余开发者完成开发，比原版游戏的体验还好。

https://www.ea.com/games/plants-vs-zombies，官方网站。

https://www.spriters-resource.com/ds_dsi/pvszds/，植物大战僵尸多媒体资源。

https://pvzcc.fandom.com/wiki/Page_of_Resources，零碎的多媒体资源。

https://github.com/ErLinErYi/PlantsVsZombies，1.1k 星，C++ 和 CMake 开发，使用 Cocos2d-X，支持多平台，游戏体验逼真，文档较多，其中多媒体资源需要联系作者获取。

https://blog.csdn.net/qq_40630246/article/details/102643196，1.1k 星项目作者的博客，给出更详细的更新日志。

https://github.com/marblexu/PythonPlantsVsZombies，2.5k 星，完全由 python 开发，游戏体验尚可，有多媒体资源。

https://github.com/chaoyang805/PlantsVsZombies，Cocos2d-x 开发，没有文档，但是给出了多媒体资源。

https://github.com/yangsheng6810/pvz，C++ 开发，给出了多媒体资源。

https://github.com/Zhuagenborn/Plants-vs.-Zombies-Online-Battle，374 星，C++ 开发，画风接近原版游戏。

1.3 王国保卫战（KingdomRush）

参考中国人大大作业的塔防游戏，是个较热门选题。该游戏界面简洁，规则有趣，有爱好者对其进行了复现，不过尚没有一个 C++ 项目随附完整的文档和多媒体资源。

https://github.com/wuhaoyu1990/KingdomRush，154 星（Github 最高），C++ 开发，采用 Cocos2dx，但是截止 2016 年就没有再维护，没有公布多媒体资源。

https://github.com/exmex/KingdomRushFrontiers，57 星，同样采用 Cocos2dx 开发（C++），公布了多媒体资源，最后一次更新截止 2019 年。

2 基本要求

在实现基础功能的前提下，参考同济大学大作业要求，制定以下基本要求。

本次项目旨在锻炼面向对象程序设计的能力，重点在于对类、继承、多态等面向对象程序设计特点以及对于复杂 C++ 类库的使用。

2.0.1 游戏引擎

统一使用 Cocos2d-x 游戏开发框架，不得只包含控制台界面。本项目偏向于锻炼代码和架构设计，请勿将重点错误地放在精美的界面上。

2.0.2 代码质量和安全

- 没有内存泄露，程序很少崩溃。
- 对于复杂逻辑使用单元测试验证正确性。
- 使用断言验证程序性质。
- 合理的异常抛出与处理。

2.0.3 代码规范

- 代码在缩进、命名等方面基本遵循了统一的风格（如 Google C++ Style）
- 正确地使用 C++ 风格类型转换（如 static_cast、dynamic_cast）且没有使用 C 风格强制转换。
- 尽可能地使用 const 和引用。

- 类的设计合理规范。

2.0.4 代码抄袭

允许借鉴开源项目代码，但应在学习他人写法的基础上，自己完成编写，且在文档中注明参考链接。

2.0.5 开发平台和 IDE

开发平台选用 Ubuntu 22.10，IDE 定为 QtCreator。

2.0.6 架构

统一使用 cmake 组织目录结构，要求项目目录结构良好、清晰。

2.0.7 C++ 新特性

项目使用 C++17 标准，要求合理地使用下列 3 条以上的 C++ 新特性。

- 初始化列表
- 类型推断（auto / decltype）
- 基于范围的 for 循环
- 智能指针
- 常量表达式（constexpr）
- Lambda 表达式
- 右值引用
- 字符串字面量
- 其他 C++11 或更高的新特性

2.0.8 C++ 功能

要求合理地使用下列 3 条以上的 C++ 功能。

- STL 容器
- 迭代器
- 类和多态
- 模板
- 异常
- 函数重载
- 运算符重载

2.0.9 版本控制与团队协作

- 项目必须使用 Git 版本控制，最终需发布在 Github，注意保留每个人的提交记录。
- 遵循 git 最佳实践（如 commit 历史干净规范、commit 描述规范、commit 包含且仅包含一项功能、使用 Pull Request 功能或分支进行团队协作等）。
- 团队成员分工需要平等、合理。

2.0.10 文档

使用 Lyx 编写项目文档。

3 功能要求

由于本项目基于 LZ 项目开发，而 LZ 项目功能比较全面，因此将在功能方面进行简化。但是为了保证项目结构的完整性，不能一味对功能进行简化，还是要适当的保留一些功能，即使用处不大。

3.0.1 已经实现的功能和界面

1. 主菜单界面、下分用户存档选择界面、用户设置界面和退出界面。游戏属性设置和用户存档功能。
2. 关卡选择界面。关卡选择功能。
3. 选择植物界面。选择植物功能，预览本关卡僵尸种类和数量功能，用户设置功能（重用）。
4. 对战界面。种植\铲除植物功能，除草机功能，僵尸死亡和关卡结束掉落金币功能，用户设置功能（重用），其他基本的对战功能。
5. 游戏结束界面。植物胜利后奖励一包金币，僵尸胜利后显示被吃掉脑子。

3.0.2 未来考虑实现的功能和界面

1. 不同关卡显示不同的背景和音乐。
2. 允许存档当前对战的状态。
3. 关卡随着通关一步步解锁，而不是一次性全部解锁。
4. 增加植物和僵尸种类。
5. 自己制作游戏素材，去除水印。

3.0.3 暂不实现的功能和界面

1. 冒险游戏下的游戏类型功能，比如通关必须保留 5 辆小车，不能让僵尸踩坏花坛等要求。
2. 彩蛋功能，该功能在 LZ 项目中存在 bug。

4 详细设计

原作者在开发本项目时，参考了很多 cocos2dx 的设计模式，但是有些应用的不彻底，不合理，我会对应进行改进。另外在编码规范作者也有不少应用不合理的地方，我在项目中一一进行了更正。最后在软件架构的设计，作者的设计其实是比较混乱的，我进行了几次大改，但核心理念不变，比如仍旧使用文件存储数据（格式包含 XML、json 等）、仍旧使用单例模式存储游戏运行过程中所需和产生的数据。

4.1 功能说明

以下是我改写后的项目功能说明，由于改写主要集中在基础类上（位于文件夹 Based 下），因此会详细介绍部分基础类的功能，并给出与 LZ 项目不同之处。其他文件夹，即文件夹 Plants、Scenes 和 Zombies，其下的类只有小幅删减而没有大的改动，因此参照 6节即可。

4.1.1 文件夹 Based

该文件夹下的类都是基础类，为上层的游戏界面类提供各种功能，包括读写数据，定义接口。

4.1.1.1 文件夹 GameData

该文件夹下的类都处理游戏相关数据。

类 GameData 保存游戏运行相关数据，如关卡数据，游戏素材路径，游戏交互文本。从 LZ 项目中的类 UserInformation 分离出来的类。

友元是 LevelDataHandler 类，该类可以访问本类所有私有成员。

类 LevelData 存放关卡数据，如僵尸数量，僵尸出现的频率，通关后奖励的金币数。相比 LZ 项目中的类 LevelData，将 readLevelData 方法转移到类 LevelDataHandler 中。

类 LevelDataHandler 采用单例模式，用于读写关卡数据。将 LZ 项目的 OpenLevelData 类更名为该类，使语义更加明确。

类 ResourcePath 存放游戏素材路径，可以自行从 XML 文件读取路径并初始化。

4.1.1.2 文件夹 UserData

类 CaveFile 存放用户存档和默认存档的相关参数。

类 UserData 保存用户相关数据，如用户设置状态，用户存档参数，用户世界数据等。从 LZ 项目中的类 UserInformation 分离出来的另一个类，和 GameData 合并后是类 UserInformation。

类 UserDataHandler 采用单例模式，用于读写和用户相关的数据（包括用户独有的关卡数据，如累计获得金币数，关卡进度）。将 LZ 项目的 UserData 类更名为该类，使语义更加明确。

类 UserSelectCard 用户选择的植物卡片编号。

类 UserSetting 存放用户设置的状态还有实时音量。

类 UserWorldData 存放用户在关卡中需要用到的数据，通关情况等。

4.1.1.3 类 AppDelegate cocos2dx 自动生成的类，该类用于启动游戏，我对该类进行了一点点修改，改动包括将设计分辨率修改为 1920*1080。

4.1.1.4 类 DataLoader 用于在启动游戏时加载用户和游戏数据。使用方式是在类 AppDelegate 的定义 DataLoader 对象，调用默认构造函数的同时就完成了数据的加载工作。

4.1.1.5 类 Runtime 采用单例模式，用于保存运行时的游戏和用户数据，便于展示游戏界面的类随时访问，该类还具有检测游戏环境的功能。

使用方式是在文件夹 proj.linux 的 main.cpp 的 main 方法中定义 Runtime 对象，定义后将会把此对象的 this 指针装载到静态指针成员中，当退出 main 函数的作用域时，静态指针也就是 this 指针将会被自动析构。

4.1.2 文件夹 Helper

4.1.2.1 类 Decryptor LZ 项目中对 Resource/resources/Text 文件夹下的部分文件进行了加密，格式一般为.reanim.compiled，需要使用项目中的代码解密。这里我提取了 LZ 项目中类 UserData 和类 OpenLevelData 的解密函数加入到该类当中用于解密，解密后的文件我都保存到原路径下，名称不变（格式遵从解密后的内容）。

4.2 内存管理

4.2.1 单例模式

对于单例模式的内存管理方式参考了两种类型，设计单例类时需要根据不同的情况进行选择：

- 一种是 cocos2dx 中 UserDefault 和 FileUtils 类的单例模式，这种单例模式获取时创建静态实例，使用完需要手动释放静态实例。注意在定义这种单例模式时，构造函数和析构函数都要定义为**私有的或保护的**（如果该类需要被继承），避免手动实例化的情况出现。
- 还有一种是 cocos2dx 中 Application 类（定义于 CCApplication-linux.h/.cpp）的单例模式，这种单例模式需要定义普通实例，使用时通过 getInstance 方法获取静态实例（其中就是普通实例，两者保持一致）。使用完无需手动释放，普通实例离开其所在作用域后，自动释放其中的静态实例。注意这种单例模式需要将构造函数定义为**公有的**，否则定义实例时不能直接访问。
- 老师在 options 库使用的单例模式，分别定义了实例类和单例类，和上一种模式的区别是无需手动释放创建的实例，只需把静态实例的指针置空即可。

4.2.2 cocos2dx 的内存管理

https://cloud.tencent.com/developer/article/1777104，在“使用 const char* 和 std::string”一节解释了哪些情况下需要释放对象。

https://forum.cocos.org/t/cocos2d-3-2/14229，cocos2d-x3.2 要不要手动删除指针的。

cocos2dx 自带内存管理的功能，和其相关的对象如果是库函数创建的，而不是 new 创建的，就不用通过 delete 释放。

4.3 游戏数据（含用户数据）

4.3.1 用户数据

4.3.1.1 垂直同步 垂直同步 (VSync) 将游戏或应用程序的图像帧速率与显示监视器的刷新速率进行同步，有助于建立稳定性。如果不同步，则可能会导致画面撕裂，即图像看起来在整个屏幕上呈现水平方向毛刺或重影的效果。

https://www.intel.cn/content/www/cn/zh/support/articles/000005552/graphics.html，英特尔介绍垂直同步技术。

4.3.1.2 音效和背景音乐的区别 背景音乐是各种音乐的一种，而音效是所有音乐大小的基础，不指代某个特定的音乐大小。

对于音乐的设置，需要重启游戏，才能让设置生效，但其他的设置都能得到实时反馈。此处的优化暂不考虑，仅使用作者之前的代码。

4.3.1.3 用户设置读写 https://blog.csdn.net/Xiejingfa/article/details/50580793，直接看对 UserDeafult 的类总结即可。

主要的用户设置通过 cocos2dx 的 UserDefault 类保存，保存的原理是对 UserDefault.xml 文件的读写。此处将 UserDefault.xml 存放在/home/gx/.config/pvz 文件夹下，该路径实际上是 cocos2dx 的 FileUtils 实例的 getWritablePath() 方法获取的。

4.3.1.4 FPS 根据机器的不同，帧率会有所变化，但是默认帧率一般在 30fps 之间，而高帧率在 60fps 之间。

4.3.2 游戏数据

4.3.2.1 分辨率 分辨率必须设定为 1920*1080，否则图片缩放会失常。调整分辨率需要在 Based/AppDelegate 下完成，添加下述代码。

```
static cocos2d::Size designResolutionSize = cocos2d::Size(1920, 1080);
```

4.3.3 暂不存储和设置部分用户和游戏数据

像是"ISBEGINSHOWEGGS"，"USEPLANTSNUMBERS"，"BREAKTHROUGH"的用户数据暂不设置和存储，看起来意义不大。

以下游戏数据也不保留，都是些次要功能。

```
bool _isEncryption;                          /* 是否加密 */
bool _zombiesIsVisible;                      /* 僵尸是否隐身 */
bool _zombiesIsSmall;                        /* 是否是小僵尸 */
bool _zombiesIsBig;                          /* 是否是巨人僵尸 */
bool _isNoPlants;                            /* 是否不可种植 */
int _flowerPosition;                         /* 花坛位置 */
int _usePlantsNumbers;                       /* 使用植物数量 */
float _userLose;                             /* 玩家失败 */
vector<MyPoint> _noPlantsPosition;           /* 不可以种植的地方 */
```

```
vector<int> _gameType;                                    /* 游戏类型 */
```

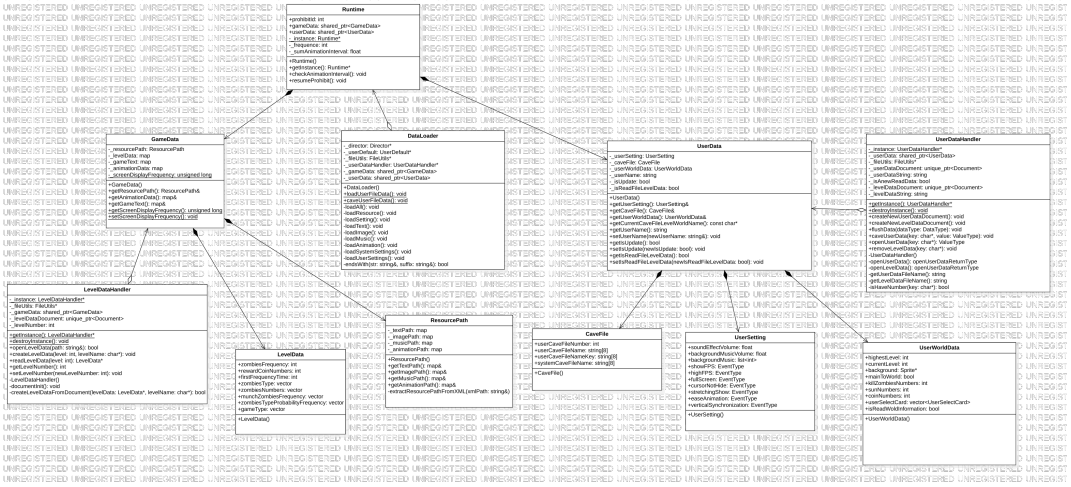
4.4 UML 图约定

UML 图中我设计的基础类将详细绘制属性、方法和可达性，而 LZ 项目中更改较少的类将不再详细绘制，仅仅给出类名。

由于 staruml 自身的原因，属性的详细类型（如是否为 const，模板参数）将被忽略，析构函数将被忽略（其访问属性保证和构造函数相同）。

为了控制 UML 图的规模，枚举类和规模很小的类不进行绘制。

最终绘制了部分重要的基础类的 UML 图，其他类的 UML 图请参见 LZ 项目，这里就不重新绘制了。



4.5 编码规范

https://zh-google-styleguide.readthedocs.io/en/latest/google-cpp-styleguide/naming/, Google 编码规范，尚未采用。

https://blog.csdn.net/K346K346/article/details/81395342，C++ 命名方式建议，有些值得参考的地方。

4.5.1 驼峰命名

https://www.zhihu.com/question/465049546，为什么 c/cpp 语言的函数好像没有按照驼峰命名法呢?

https://www.ruanyifeng.com/blog/2017/02/filename-should-be-lowercase.html，为什么文件名要小写?

历史原因，C/C++ 的早期代码往往全部小写，并且大量使用缩写，这对于项目的维护不太方便。我采用 Java 中广泛应用的驼峰书写，而对于类名上也和 Java 保持一致，类名即文件名，提高辨识度。老师在开发彭水项目时，使用的也是这种命名规范，因此准备沿用下去。

另外英文单词能拼写全尽量拼写全，长点没关系，不要使用缩写，要不会提高别人理解代码的难度。

4.5.2 成员变量前加下划线

https://cloud.tencent.com/developer/article/1805416，C++ 变量前面加下划线的含义。

cocos2dx 类中的私有成员变量前都加上了下划线，遵从了 Obejective-C 实例变量定义的风格。暂不评论这种风格应用在 C++ 中的好坏，为了减少迁移代码的工作量，决定所有**私有**成员变量前也都加上下划线（公

有成员变量不受影响），和 coco2dx 的编程风格保持一致。需要指出，Alt+Enter 生成 Getter 和 Setter 方法时不会受前置下划线影响。

4.5.3 不使用全局变量

https://zhuanlan.zhihu.com/p/59636541，使用全局变量会带来诸多问题。

4.5.4 =default 的使用

由于构造函数的实现往往在.cpp 文件中，其中使用了初始化参数列表。但是初始化参数列表在头文件的构造函数声明中不会体现，因此决定使用 =default 关键字区别默认实现和其他实现。

4.5.5 尖括号与双引号包含区别

为了和我编写的代码文件区别开来，游戏库文件和系统库文件统一使用尖括号包含，而我编写的代码文件使用双引号包含，如下所示。

```
#include <cocos2d.h>
#include <ui/CocosGUI.h>

#include <map>
#include <string>

#include "Based/Runtime.h"
```

4.5.6 单入单出

按照编码规范，函数应该只有一处返回值，如果有多种（或多个）返回值，可以考虑使用枚举或者结构体封装。LZ 项目中许多代码虽然使用了枚举封装，但是返回值的地方有多处，不符合单入单出的规范，由于涉及代码数量过多因此不做更改。

4.5.7 switch 中部分 case 使用花括号包裹

https://stackoverflow.com/questions/61708267/jump-bypasses-variable-initialization-in-switch-statement，Jump bypasses variable initialization in switch statement。

switch 中某个 case 下如果定义变量或者符号应使用花括号包裹，可以把作用域控制在当前 case 下，而不会影响之后的 case。

4.5.8 函数重载和函数模板

函数重载主要针对参数个数不同的情况，而函数模板针对参数类型不同，但个数相同的情况。相比之下，函数模板更具有广泛性。另外函数模板可以重载，但是类模板不能重载。

4.5.9 getter 和 setter 方法的约定

get 方法一般仅用来获取值，不用来更改值，因此返回变量的拷贝，不返回引用。但是有时需要返回对象成员并对其下特定成员进行更改，这时需要返回该对象引用。

4.5.10 类的成员变量应该使用引用类型、指针类型还是对应类型对象

https://blog.csdn.net/bumingchun/article/details/112755993，简洁的给出该问题的定义和结论，一般没有特殊需求，成员定义为对应类型对象是最好的。

4.5.11 结构体和类的选择

如果某对象仅组合在其他的类中作为成员，有 getter 和 setter 方法的接口，那么该对象的类定义为结构体即可。无需层层封装 getter 和 setter 接口。

4.5.12 枚举名约定

https://blog.csdn.net/craftsman1970/article/details/106026360，不使用全部大写的枚举名。

为了避免不必要的编码错误，不使用汉字作为枚举名。为了避免和宏冲突，不使用全部大写的枚举名。此处使用首字母大写，其他字母小写的枚举名。

4.5.13 .cpp 需要的头文件要转移到.h 文件中吗

不需要。#include 尽量写到 cpp 文件里，因为两个文件在.h 文件里相互 include，就会产生编译错误，而两个文件在.h 文件互相 include，就不会有该问题，因此在.h 文件 include 就要避免互相包含的问题，而.cpp 文件就不需要考虑。

4.6 使用 C++17 标准

/home/gx/Documents/pvz/pvz/cocos2d/cmake/Modules/CocosConfigDefine.cmake,其中定义了 cocos2d 项目使用的 C++ 标准，默认为 11，这里我尝试改成了 C++17，可以运行，所以就用 C++17，享受更多语法特性带来的便利。注意：仅仅在顶层 CMakeLists.txt 的更改 C++ 标准是不够的。

```
# check c++ standard
set(CMAKE_C_STANDARD 99)
set(CMAKE_C_STANDARD_REQUIRED ON)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

https://stackoverflow.com/questions/67432538/cannot-set-cmake-warn-deprecated-inside-the-cmakelists-txt，使用 CMake 命令消除 deprecated warnings。

更改为 C++17 标准后，第一次构建整个项目时会出现了大量的 deprecated warnings，如下所示。但是之后不会反复弹出，所以暂不考虑设定特别的 CMake 选项去消除它。

```
warning: ‘template<class _Category, class _Tp, class _Distance, class _Pointer, class _
```

4.7 部分问题解决方案

4.7.1 语法相关问题

本节记录项目相关的部分语法问题，还有一部分语法问题的解决方案已经迁移到 C++ 专题笔记下。

4.7.1.1 non-const lvalue reference type __normal_iterator<> cannot bind a temporary of type __normal iterator in gcc https://stackoverflow.com/questions/58867984/non-const-lvalue-reference-type-normal-iterator-cannot-bind-a-temporary-of-ty

4.7.1.2 warning: object backing the pointer will be destroyed at the end of the full-expression for std::pair https://stackoverflow.com/questions/64657316/warning-object-backing-the-pointer-will-be-destroyed-at-the-end-of-the-full-exp

4.7.1.3 libfmod.so: .dynsym local symbol at index 2 (>= sh_info of 2) 暂时使用在 CMakeLists.txt 中添加-fuse-ld=gold 解决，但是还是不管用。

https://stackoverflow.com/questions/59915966/unknown-gcc-linker-error-but-builds-sucessfully，提出使用-fuse-ld=gold 选项，但是貌似不管用。

4.7.1.4 ISO C++11 does not allow conversion from string literal to 'char *' 将转换的目标变量设置为 const char* 类型即可。

4.7.1.5 Cannot initialize object parameter of type 'cocos2d::Ref' with an expression of type 'MainMenu' 在 MainMenu 中，由于使用到的 ui 类型定义没有覆盖全，只需包含所需头文件即可解决此类问题。

```
#include <ui/CocosGUI.h>
using namespace cocos2d::ui;
```

4.7.1.6 Non-const lvalue reference to type 'cocos2d::Vec2' cannot bind to a temporary of type 'cocos2d::Vec2' https://blog.csdn.net/jokerMingge/article/details/121554826，“非常量引用的初始值必须为左值“及“匿名对象“。

诸如此类问题其实是匿名对象导致的，因为匿名对象是 prvalue(纯右值)，无法作为引用传递，若想引用参数接收匿名对象只有用 reference-to-const 接收。当一个对象被用作右值时，用的是对象的值，当对象被用作左值的时候，用的是对象的身份。而引用只能引用身份，不能引用纯值（如匿名对象）。

4.7.1.7 delete 野指针 https://www.zhihu.com/question/38998078，C++ 里 delete 指针两次会怎么样?

另外需要指出，delete 一个指针之后，只是回收该指针指向的堆空间，而指针本身的值不变（也就是说 delete 并不将指针删除，同时指针仍然指向原来的那个地址）。因此 delete 后将指针置空是个好习惯，因为 delete 空指针是合法的，没有副作用。

4.7.1.8 多用户存档 https://rapidjson.org/classrapidjson_1_1_generic_document.html,GenericDocument 类的官方文档。

暂不支持多用户存档，一个原因是这是次要功能，另外一个原因是在实现过程中发现 UserDataHandler 的 createNewUserDataDocument() 方法，反复调用可能会出现问题，string 类型和 rapidJson::GenericDocument 释放过程中会崩溃。

以上问题已解决，是打开和保存用户数据的方法定义不当，使用函数模板和 if constexpr 语法后解决了问题。

4.7.1.9 模板返回类型的默认值 https://stackoverflow.com/questions/38038860/default-or-zero-value-of-unknown-template-type，Default or zero value of unknown template type，给出的解决办法简单合理。

https://stackoverflow.com/questions/12615549/how-to-get-the-default-value-of-any-type, How to get the default value of any type，讨论了多种解决办法，但是结果不太清晰。

有模板返回类型默认值的需求，但是 C++20 前还没有很好实现的办法，以下是 workaround。考虑到兼容和语义清晰，我选择 return T()。

```
template <typename T>
T void fun() {
        // ...
        return T() or T{} or {};          // use {} since C++11
}
```

4.7.1.10 函数模板声明与实现分离 https://www.zhihu.com/question/535308979/answer/2512192023，最高赞回答解释的非常清晰。

https://blog.csdn.net/jslove1997/article/details/118339198，简单解释了不能分离定义的原因。

在 C++20 之前不建议将函数模板声明与实现分离在不同文件中，因为语法和编译器支持尚不到位。

4.7.1.11 根据不同的模板参数完成对应操作 https://stackoverflow.com/questions/11021267/perform-different-methods-based-on-template-variable-type，Perform different methods based on template variable type，提问者重现了我的问题。

https://stackoverflow.com/questions/73330007/c-conditionally-call-functions-based-on-type-of-the-template-parameter，提问者很好地描述了我的需求，在 C++17 下有解决方案。

虽然有 workaround 方法，但建议使用函数重载。

4.7.1.12 FMOD error! (51) Error initializing output device. https://github.com/cocos2d/cocos2d-x/issues/16627，提出两种解决方案，一种是更改 AudioEngine 的代码，另外一种是安装 pulseaudio，采纳了后者。

可能是使用 C++17 编译导致的问题，在安装 pulseaudio 之后解决了，安装命令如下所示。

```
sudo apt install pulseaudio
```

部分错误信息如下所示。

```
FMOD error! (51) Error initializing output device.
FMOD error! (67) This command failed because System::init or System::setDriver was not c
sound effect in resources/Music/Clocks.ogg could not be preload
...
```

4.7.1.13 auto 并不会自动引用 https://blog.51cto.com/luweir/4875857, 在 std::vector<int> 上测试 auto 需不需要加引用符。

如果 auto 所代表的类型需要引用的话，需要显式加上引用符 &。

4.7.1.14 const char* 和 char* 作为返回值的问题 https://bbs.csdn.net/topics/50246945，关于 const char* 和 char* 作返回值的讨论。

在使用 const char* 之前，首先要了解 const char* 的含义，以及与 char* const 的区别，如下代码所示。

一般 const char* 指向的字符串所占用的内存大小都是固定的，因此修改其值有可能导致数组越界，基于此返回 const char* 避免指针所指向的值被意外更改。

```
char* const str1 = "Hello, World";        //常指针，指向字符串
*str1 = 'M';                                        //可以改变字符串内容
str1 = "Bye, World";                              //错误，如能改变常指针指向的内存地址

const char* str2 = "Hello, World";        //指针，指向字符串常量
*str2 = 'M';                                        //错误，不能改变字符串内容
str2 = "Bye, World";                             //修改指针使其指向另一个字符串
*str2 = 'M';                                        //错误，仍不能改变字符串内容
```

4.7.1.15 对引用取地址 https://bbs.csdn.net/topics/350058236，讨论区给出了回答，引用就是别名，他们的地址一样!。

4.7.1.16 Lambda 使用 [&] 捕获时要格外小心 Lambda 使用 [&] 捕获时要格外小心，如果捕获的是基础类型值，很可能在 Lambda 函数执行时已经被释放了，此时再访问就相当于访问野指针，如下代码所示，之前通过 [&] 捕获的 ID 在 Lambda 函数执行时已经被释放，这里更正为 [=] 拷贝捕获。

另外在使用引用捕获时还遇到了其他问题，尚未找出原因，建议能使用拷贝捕获就使用拷贝捕获。

```
void InputDataMenu::createButtons(Sprite* sprite, const std::string& Label, const Vec2&
                                  const float scale, const int ID) {
    auto button = Button::create("button.png", "button_down.png", "", TextureResType::PL
    button->setPosition(vec2);
    button->setScaleX(scale);
    button->setTitleLabel(label(Label, 20, Vec2(0, 0), 0, Color3B::GREEN, 1.0f / scale))
    sprite->addChild(button);

    CaveFile* caveFile = &(_runtime->userData->getCaveFile());

    button->addTouchEventListener([=](Ref* sender, Widget::TouchEventType type) {
        switch (type) {
            case Widget::TouchEventType::BEGAN: {
                cocos2d::log("Widget::TouchEventType::BEGAN, ID: %d", ID);
                MusicPlayer::playMusic("gravebutton");
                break;
```

```
        }

        case Widget::TouchEventType::ENDED: {
            cocos2d::log("Widget::TouchEventType::ENDED, ID: %d", ID);
            switch (ID) {
                case 1: { /* 确定 */
                    if (!_textField->getString().empty() &&
                        _textField->getString() != "未命名存档") {
                        caveData();
                        updateButtonText();
                        this->removeChildByName("_shieldDialogLayer");
                        _inputString.clear();
                    }
                    break;
                }
                case 2: { /* 取消 */
                    if (_runtime->userData->getUserName() != "未命名存档") {
                        this->removeChildByName("_shieldDialogLayer");
                        _inputString.clear();
                    }
                    break;
                }

                case 3: {
                    _userDefault->setIntegerForKey("USERDATANUMBER",
                                                   _caveFileNumber); /* 记录所选存档
                    caveFile->userCaveFileNumber = _caveFileNumber;
                    UserDataHandler::getInstance()->createNewUserDataDocument();
                    DataLoader::loadUserFileData();

                    const int id = caveFile->userCaveFileNumber;
                    const char* key = caveFile->userCaveFileNameKey[id].c_str();
                    string str = _userDefault->getStringForKey(key);

                    /* 读取所选存档的名字并更新 */
                    if (!str.empty()) {
                        _runtime->userData->setUserName(str);
                    } else {
                        auto userCaveFileName = caveFile->userCaveFileName;
                        string userName = userCaveFileName[_caveFileNumber];
                        _runtime->userData->setUserName(userName);
                    }
```

```
                        _runtime->userData->setIsUpdate(true);

                        deleteDialog();

                        Director::getInstance()->replaceScene(MainMenu::create());
                        break;
                    }

                    case 4: deleteDialog(); break;

                    case 5: createShieldLayer(); break;
                }
            }

            default: {
                cocos2d::log("Enumeration values 'MOVED' and 'CANCELED' not handled in s
                             "InputDataMenu::createButtons");
                break;
            }
        }
    });
}
```

4.7.1.17 野指针 迁移项目时注意野指针问题，要在构造函数或者其他位置对指针赋值或者置空。

4.7.1.18 在派生类的构造函数中初始化基类成员 派生类的构造函数的初始化列表可以包含基类的构造函数、派生类成员的初始化，但是不能有基类成员的初始化！但是可以在派生类构造函数体中初始化基类成员，但是不建议这么做。派生类构造函数一般形式如下。

```
派生类构造函数名(总参数表列): 基类构造函数名(参数表列), 子对象名(参数表列)
{
        派生类中新增数成员据成员初始化语句
}
```

本项目仍旧选择在派生类构造函数体中初始化基类成员，仅仅是为了方便和直观，后续考虑按照上述一般形式调整。

4.7.2 IDE 相关问题

4.7.2.1 rename symbol under cursor 不可靠 使用该功能，有时会改出多余的符号，这时需要手动排查。

4.7.3 系统相关问题

4.7.3.1 垂直同步 https://stackoverflow.com/questions/62023435/do-you-have-to-call-glfwswapinterval1-in-an-opengl-program，Do you have to call glfwSwapInterval(1) in an OpenGL program?

将 wglSwapIntervalEXT 替换为 glfwSwapInterval 似乎能达到同样的目的。

4.7.3.2 同时播放音乐数量的限制 https://github.com/cocos2d/cocos2d-x/issues/17151，Linux 下播放音乐超出限制后无法再播放新的音乐，目前在 cocos2dx 的 3.15 版本还没有修复。

https://discuss.cocos2d-x.org/t/fail-to-play-cause-by-limited-max-instance-of-audioengine/26800/6,不建议使用 setMaxAudioInstance 来突破音乐播放数量的限制，一种解决办法是手动移除历史播放的音乐，但是非常麻烦。

4.7.3.3 新增游戏音乐 https://convertio.co/zh/mp3-ogg/，mp3 转换为 ogg 格式。

如果在 resources/Music 下新增了音乐文件，需要将路径添加到 resources/Text/MusicPath.xml 下，代码如下所示。为了触发 MusicPath.xml 的重新加载，需要将存放用户数据的文件夹删除，路径为/home/用户名/.config/pvz，参见 4.3.1.3节。

最后音乐格式需要转换为 ogg 格式。

```
<p secretlabs="resources/Music/secretlabs.ogg"/>
```

4.8 基于 apache license 2.0 项目二次开发后再开源

https://github.com/ErLinErYi/PlantsVsZombies/blob/master/LICENSE,LZ 项目采用协议的英文介绍。

https://choosealicense.com/，Github 官方提供选择协议的指导。

https://www.v2ex.com/t/881326，指出代码声明里保持他们的声明就可以，不要用他们的商标宣传。

https://segmentfault.com/a/1190000022973105，当使用声明了 Apache License Version 2.0 的软件时，需要有显著的声明，不得有隐瞒和可以忽略的色彩。

LZ 项目采用的是 apache license 2.0，我基于 LZ 项目二次开发再开源需要遵循该协议。

5 LZ 项目配置

LZ 项目——在本文档指代 ErLinErYi 开发的植物大战僵尸项目，以下是 GitHub 项目和资源文件下载路径。

https://github.com/ErLinErYi/PlantsVsZombies，学习项目地址。

https://caiyun.139.com/m/i?135ClQmyJEXz2，该项目资源文件下载。

由于 LZ 项目是使用 Visual Studio 搭建的，因此准备在 VS 下搭建并学习源码，再迁移到 Linux 开发。

该项目执行 ISO C++14 标准，在 Windows 下开发，之前尝试在 Linux 下搭建，尚未成功。

5.1 IDE

5.1.1 Visual Studio

经过尝试，成功在 Visual Studio 2019 和 Visual Studio 2017（vs2019）下搭建该项目。

在 Visual Studio 2017 下，如果 Windows SDK Version 配置不正确的话，会报编译器内部错误（An internal error has occurred in the compiler）。

在 Visual Studio 2019 下，编译时出现宏 CC_FORMAT_PRINTF 未定义的问题，位于 libcocos2d 下的 CCPlatformMacros.h，表面的原因是代码本身的逻辑判断跳过了对 CC_FORMAT_PRINTF 的定义，但实际上还和工具链有关，该问题尚未发现不利影响。

在 Visual Studio 2020 下编译报标识符不允许中文问题（this character is not allowed in an identifier），在此之前由于中文乱码，将项目文件全部转码为 UTF-8，怀疑是此项操作导致的问题，后续未尝试。

https://visualstudio.microsoft.com/zh-hans/vs/older-downloads/，下载 vs2017 企业版，版本 15.9（激活前可以试用 30 天）。

安装时选中 Desktop development with C++，采用默认选项的同时，注意增加选项 Windows 8.1 SDK。

克隆 LZ 项目，将资源文件放置在指定目录，然后使用 vs2017 打开根目录下的 PlantsVsZombies.sln。

5.1.1.1 VS 配置 将 Solution Platforms 设置为 x86。

右击 PlantsVsZombies、libSpine 子项目，点击 Properties 打开，将 Platform Toolset 设置为 v141，Windows SDK Version 设置为 10.0.17763.0（或其他 10.0 的 SDK）。

右击 librecast、libcocos2d、libbox2d，点击 Properties 打开，将 Platform Toolset 设置为 v141，Windows SDK Version 设置为 8.1。

5.1.1.2 缺少 MSVCR110.dll 通过安装 x86 版本的 Visual C++ Redistributable for Visual Studio 2012 解决问题。

https://www.microsoft.com/zh-CN/download/details.aspx?id=30679，Visual C++ Redistributable for Visual Studio 2012 下载地址。

https://www.drivereasy.com/knowledge/how-to-fix-msvcr110-dll-is-missing/，给出 5 种解决方案，依次递进。

https://blog.csdn.net/lemonisan/article/details/109146494，来自中文社区，给出三步解决方案，依次递进。

5.1.1.3 修改语言包(显示语言) 我更习惯用英文，修改方法参见：https://blog.csdn.net/Bob___yuan/article/details/106

5.1.1.4 Debug 信息格式设置（Error C3130） https://discuss.cocos2d-x.org/t/failed-to-write-injected-code-block-to-pdb/12519/12，讨论的最后给出了解决方案。

使用 Debug 模式编译项目时出现 Error C3130 Internal Compiler Error: failed to write injected code block to PDB。

Project->Properties->Configuration Properties->C/C++->General->Debug information format->C7 compatible(/Z7)

5.2 中文乱码

通过 Visual Studio 2019 启动似乎解决了乱码问题，但还是要说明下乱码原因，源码文件大部分以 GBK（即 GB2312）编码，而从 Visual Studio 2022 打开时按 UTF-8 解码，所以出现了乱码。

解决乱码的办法可以通过转码完成，不过在项目中引入了新的问题，因此不建议采用。

在 Windows 平台下我找到软件 CodeTramit 可以实现批量转码,使用方便,下载地址:https://blog.csdn.net/dpsying/a
也可以直接使用 Visual Studio 修改当前页面编码(不能批量处理),参见:https://blog.csdn.net/weixin_42109012/artic
在 Linux 平台下考虑使用 iconv 完成文本内容的转码，不过该命令（iconv）一次只能转码一个文件，需要配合 Shell 脚本批量读入。

5.3 代码风格

5.3.1 宏 ___MAIN_H___

宏 ___MAIN_H___ 在 main.h 中使用，LZ 项目中的 main.h 对应 PlantsVsZombies.h。
https://bbs.csdn.net/topics/250073215，main.h 的用法。

main.h 可以作为一个公共的头文件，包含一些公共头文件，声明一些自定义数据类型，自定义常量什么的，这些信息放在各自 c 文件的头文件中也不太合适，可以在 main.h 中声明。当然，也不是一定必须的，也可以写一个专门定义常量、类型的头文件之类的。这样做，总览的逻辑关系清晰一点。

5.4 多媒体文件

5.4.1 动画文件

目前给出的资源文件中的动画文件是编译过的。
https://plantsvszombies.fandom.com/wiki/Modify_Plants_vs._Zombies，动画文件的修改。
https://www.bilibili.com/read/cv5476547/，以向日葵的动画修改为例。
https://plantsvszombies.fandom.com/f/p/3418359504830875373,提供了 PVZUtil 1.0 提取.reanim.compiled 格式的动画文件，但是不适用于.reanim.compiled 格式的 xml 文件的提取。

5.4.2 精灵图集

一个 cocos2dx 的精灵图集由后缀.plist 和.pvr.ccz 构成，其中 plist 文件指明了图集中每张图片的名字和在图集中的位置，pvr.ccz 文件是对应的图集，可以使用 TexturePacker 预览，也可以使用该软件另存为.png 格式的图片。目前尚没有找到一个软件可以实时预览图集中每张图片的名字和区域，不过可以使用脚本将图集中每张图片中提取出来，从而得知每张图片对应的名字。
https://github.com/song0071000/UnpackPlistPng，提取图集中图片的脚本，安装限制多。
https://github.com/onepill/texture_unpacker_scirpt，提取 Texture 制作图集中的图片，兼容性好。
https://blog.csdn.net/baidu20008/article/details/41445963，提取 plist 文件中的单图或分解 plist 大图为小图。

此处选用 texture_unpacker_scirpt，在 Windows 平台的配置步骤如下，先安装 Python3.10，再安装 Pillow，脚本如下。

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
```

克隆 texture_unpacker_scirpt 到指定文件夹，将要提取的图集（即.plist 和.png 文件）放在同一文件夹下，执行下述命令提取。
需要补充的是，.png 文件是通过 texturePacker 打开.pvr.ccz 后另存为的图片。

```
python3 unpacker.py World_1
```

5.5 数据文件的加密与解密

如果增删了数据文件，那么需要对项目重新运行 cmake 文件进行同步，否则会报错说找不到已经删除的文件。

5.5.1 解密

5.5.1.1 解密函数 LZ 对 XML 文件的内容进行了加密，解密是通过以下代码完成的，其中 textpath 是加密后 xml 文件的路径，passWords 是解密的文件内容。准备通过打印 passWords 的方式获取原始文件内容，然后单独保存文件。

```
OpenLevelData::getInstance()->decrypt(textpath, passWords);
```

5.5.1.2 输出解密后的文件

```
#include <iostream>
#include <fstream>

/* 解密 */
decrypt(str, passWords);

cocos2d::log("decrypt: %s", passWords);
ofstream os;        //创建一个文件输出流对象
os.open("C:\\Users\\xigon\\Documents\\GitHub\\PlantsVsZombies\\LevelData.txt");//将对象与
os << string(passWords);    //将输入的内容放入txt文件
os.close();
```

5.5.2 正在处理的加密文件

https://jsonformatter.org/xml-parser，通过 xml-parser 可以在线读取.reanim1.compiled 格式的文件。
https://www.w3cschool.cn/tools/index?name=xmljson，在线完成 xml 和 json 的相互转换。

resources/Text/GAMEWORLD_1DATAS.reanim.compiled
resources/Text/GAMEWORLD_1DATAS_DIF.reanim.compiled
以上两个文件都经过了加密。

resources/Text/GAMEWORLD_1DATAS.reanim1.compiled
resources/Text/GAMEWORLD_1DATAS_DIF.reanim1.compiled
以上两个文件是解密后的文件，但仍需要 XML Parser 读取。

Windows 下 cocos2d 的 SAXParser 无法读取上述文件，但是也不报错，程序直接崩溃了，目前尚未修复。Linux 下也无法读取上述文件，但是有报错 cocos2d: SAXParser: Error parsing xml: expected < at。

后来发现上述文件实际上是 json 格式，需要使用 OpenLevelData 类（在 Based/LevelData.h 下定义）中读取，其中用到了 rapidjson。

5.6 依赖

5.6.1 Spine

Spine 是一款针对游戏的 2D 骨骼动画编辑工具。Spine 旨在提供更高效和简洁的工作流程，以创建游戏所需的动画。Spine 支持很多游戏工具，包括 Cocos2dx。官方的运行库可以在 GitHub 下载，其中许可证授权在你的游戏中可以使用运行库。并且所有的代码都是开源的，这些开源的功能在你的游戏中都是必不可少的。

https://github.com/EsotericSoftware/spine-runtimes/，官方运行库。

5.7 Windows API

5.7.1 使用 ___declspec(dllexport) 从 DLL 导出

https://docs.microsoft.com/zh-cn/cpp/build/exporting-from-a-dll-using-declspec-dllexport?view=msvc-170，官方文献。

https://www.jianshu.com/p/ea45468f25f1，关于 dllexport 部分解释得不够清晰。

___declspec(dllexport) 关键字可以从 DLL 中导出数据、函数、类或类成员函数。___declspec(dllexport) 将导出指令添加到对象文件中，因此你不需要使用.def 文件。

若要导出函数，___declspec(dllexport) 关键字必须出现在调用约定关键字的左侧（如果指定了关键字的话）。

```
__declspec(dllexport) void __cdecl Function1(void);
```

若要导出类中的所有公共数据成员和成员函数，该关键字必须出现在类名的左侧。

```
class __declspec(dllexport) CExampleExport : public CObject
{ ... class definition ... };
```

5.7.2 主函数 WinMain

https://developer.aliyun.com/article/246307，主函数 main WinMain _tmain _tWinMain 的区别。

https://docs.microsoft.com/en-us/windows/win32/learnwin32/winmain--the-application-entry-point，官方文档。

main 是 C/C++ 的标准入口函数名，WinMain 是 windows API 窗体程序的入口函数，而 _tWinMain 是 Unicode 版本的 wWinMain 函数别名。

<tchar.h> 中有如下几行：

```
#ifdef _UNICODE

#define _tmain        wmain
#define _tWinMain     wWinMain

#else    /* ndef _UNICODE */

#define _tmain        main
#define _tWinMain     WinMain
```

```
#endif
```

5.7.3 函数 CreateMutexW

https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createmutexw，官方文献。

5.7.4 宏 UNREFERENCED_PARAMETER

https://stackoverflow.com/questions/17192224/using-unreferenced-parameter-macro，指出 C++17 的新特性可以作为代替。

5.7.5 函数 OutputDebugString

https://cloud.tencent.com/developer/ask/sof/149656，解答 OutputDebugString 使用中的问题。

适用于 Visual Studio 输出 Debug 字符串，注意不能输出 std::string，而要使用 std::wstring。
若要输出 std::string，考虑 OutputDebugStringA()。

```
std::wstring line = "bla";
OutputDebugString( line.c_str() );
```

5.7.6 DWORD

双字节类型（int 代表一个字节）。

```
typedef unsigned long       DWORD;
```

5.7.7 DEVMODE

DEVMODE 数据结构中包含了有关设备初始化和打印机环境的信息。

```
typedef DEVMODEW DEVMODE;


typedef struct _devicemode {
  TCHAR dmDeviceName[CCHDEVICENAME]; //打印机（显示设备）名称
  WORD  dmSpecVersion;
  WORD  dmDriverVersion;  //驱动版本号
  WORD  dmSize; //结构体大小
  WORD  dmDriverExtra;
  DWORD dmFields;
   union {
     struct {
           short dmOrientation; //DMORIENT_PORTRAIT (1) 纵向 or DMORIENT_LANDSCAPE (2) 横
           short dmPaperSize; //打印纸张类型  常用 A3:8,A4,9 ,详见下图
           short dmPaperLength; //只针对打印机， 覆盖dmPaperSize指定的纸张长度 ，单位0.1m
```

```
            short dmPaperWidth; //只针对打印机，覆盖dmPaperSize指定的纸张宽度，单位0.1mm
            short dmScale; // 设置打印输出缩放因子，缩放比例为dmScale/100
            short dmCopies; //设置打印份数（如果打印机支持的话）
            short dmDefaultSource;// 指定打印机纸张来源，通过DC_BINS 标志调用DeviceCapabi
            short dmPrintQuality;//指定打印机质量（分辨率）DMRES_DRAFT(-1),DMRES_LOW(-2),D
            };
    struct {
            POINTL dmPosition;
            DWORD  dmDisplayOrientation;
            DWORD  dmDisplayFixedOutput;
            };
        };

    short dmColor; //黑白，彩色设定 DMCOLOR_MONOCHROME（1）：黑白， DMCOLOR_COLOR（2）：彩
    short dmDuplex;//单面/双面设置，DMDUP_SIMPLEX（1），单面；DMDUP_VERTICAL（2），长边装订
    short dmYResolution;//指定Y轴DPI，若初始化设定这个值，PrintQuality 设置值为X轴DPI
    short dmTTOption;
    short dmCollate;//设定打印多页时，是否整理。DMCOLLATE_TRUE（1），整理；DMCOLLATE_FALSE
    TCHAR dmFormName[CCHFORMNAME]; //表单名称，如"Letter" or "Legal"，完整表单列表通过，En
    WORD  dmLogPixels;
    DWORD dmBitsPerPel;
    DWORD dmPelsWidth;
    DWORD dmPelsHeight;

    union {
      DWORD dmDisplayFlags;
      DWORD dmNup;
          };

    DWORD dmDisplayFrequency;
#if (WINVER >= 0x0400)
    DWORD dmICMMethod;
    DWORD dmICMIntent;
    DWORD dmMediaType;
    DWORD dmDitherType;
    DWORD dmReserved1;
    DWORD dmReserved2;
#if (WINVER >= 0x0500) || (_WIN32_WINNT >= 0x0400)
    DWORD dmPanningWidth;
    DWORD dmPanningHeight;
#endif
#endif
} DEVMODEW;
```

6 LZ 项目功能说明

在学习 LZ 项目过程中一个简单的记录，便于二次开发时参考。

6.1 文件夹 Based

6.1.1 AppDelegate.h/.cpp

创建 cocos2dx 游戏项目默认生成的启动文件，运行游戏时将会执行 applicationDidFinishLaunching() 方法。

6.1.2 Dialog.h/.cpp

定义类 Dialog。定义所有对话框的接口和共有的成员变量，是一个抽象类。切换用户存档，用户设置，退出游戏等对话框都继承该类。

6.1.3 GlobalVariable.h/.cpp

定义类 Global，采用单例模式，用于保存运行游戏所需的用户数据和检查游戏运行环境。

6.1.4 LevelData.h/.cpp

用于存储和读写关卡数据，其中定义类 OpenLevelData 和类 LevelData。

- 类 OpenLevelData 用于读取关卡数据，其中包含解密函数（通过简单的加减、移位、按位与操作实现，未来将移除）。
- 类 LevelData 用于存放关卡数据。

6.1.5 PlayMusic.h/.cpp

定义类 PlayMusic，用于播放游戏音乐，其中的方法都是公有静态方法。

6.1.6 UserData.h/.cpp

定义类 UserData，用于读写用户数据，然后存储在 UserInformation 对象下。

6.1.7 UserInformation.h/.cpp

定义类 UserInformation，枚举类 WorldName，结构体 UserSelectCard 和 WorldData。类 UserInformation 用于存储所有用户和游戏数据。

6.2 文件夹 Scenes

6.2.1 文件夹 LoadingScene

6.2.1.1 LoadingScene.h/.cpp 定义类 LoadingScene，加载界面，但是该类还包含了加载用户和游戏数据的功能。

方法 openResourcesPath 打开给定路径下，包含资源文件路径的 xml 文件，通过读取 xml 文件中的路径，计算文件个数，同时将路径设置到变量中，供今后加载资源使用。其中包含字符串的加密和解密，尚不清楚为什么要增加这一步骤。

一个例子。

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <p PlantsText="resources/Text/PlantsText.xml"/>
  <p GameText="resources/Text/GameText.xml"/>
  <p GAMEWORLD_1DATAS="resources/Text/GAMEWORLD_1DATAS.reanim.compiled"/>
  <p GAMEWORLD_1DATAS_DIF="resources/Text/GAMEWORLD_1DATAS_DIF.reanim.compiled"/>
</data>
```

通过库 tinyxml2 解析后（经过整理）。

```
Name: PlantsText, Value: resources/Text/PlantsText.xml

Name: GameText, Value: resources/Text/GameText.xml

Name: GAMEWORLD_1DATAS, Value: resources/Text/GAMEWORLD_1DATAS.reanim.compiled
```

6.2.2 文件夹 MainMenuScene

6.2.2.1 InputDataScene.h/.cpp 定义类 InputDataMenu，用于选择用户存档。

6.2.2.2 MainMenu.h/.cpp 定义类 MainMenu。加载界面后进入的主菜单，该界面可以选择存档或者进入不同模式的游戏。

一个需要注意的地方是如何实现用户存档功能，原始信息是从 XML 中读取解析的，之后用户自定义的信息如何存放尚不得知。

该界面可以跳转到 OptionScene（类 OptionsMenu），HelpScene、QuitScene（类 QuitMenu）、InputDataScene（类 InputDataMenu）。

6.2.2.3 OptionScene.h/.cpp 定义类 OptionsMenu，用户可以通过该类创建的对话框设置音乐大小，是否全屏等。

6.2.2.4 QuitScene.h/.cpp 定义类 QuitMenu，退出游戏的对话框。

6.2.3 文件夹 HelpScene

6.2.3.1 HelpScene.h/.cpp 定义类 HelpScene，展示本项目相关信息。

6.2.4 文件夹 WorldScene

6.2.4.1 World_1.h/.cpp 定义类 World_1。现代世界关卡。选择关卡后，进入选择植物界面，即类 SelectPlantsScene。

_caveFileNumber 用户文档编号，是全局唯一的，默认从 0 开始计数。

readWorldLevel() 读取关卡数据信息。

GAMEWORLD_1DATAS_DIF 噩梦模式关卡数据

GAMEWORLD_1DATAS 简单模式关卡数据

6.2.4.2 SelectWorldScene.h/.cpp 定义类 SelectWorldScene，用于选择世界，比如现代世界，暗黑时代。
简单起见，只保留现代世界，而该类即将被移除。

6.2.5 文件夹 GameScene

6.2.5.1 GSBackgroundLayer.h/.cpp 定义类 GSBackgroundLayer，进入关卡后设置背景。

6.2.5.2 OpenCaveGameScene.h/.cpp 定义类 OpenCaveGameScene，如果之前保存游戏进度的话，则会读取并回到暂停场景，即当前场景。
简单起见，准备废弃该类，不保存游戏运行暂停的数据。

6.2.5.3 GSAnimationLayer.h/.cpp 定义类 GSAnimationLayer，创建地图随机生成阳光和除草机，另外控制僵尸、植物、子弹、小车事件更新。

6.2.5.4 GSInformationLayer.h/.cpp 定义类 GSInformationLayer，展示当前是第几天的冒险之旅、僵尸进攻进度条、死亡僵尸数量、金币数量、阳光数量。

6.2.5.5 GSControlLayer.h/.cpp 定义类 GSControlLayer，设置 schedule 函数循环判断控制卡片是否可以选择、植物种植位置是否合法、僵尸的文字显示与旗子更新、关卡是否结束（调用类 GSGameResultJudgement 完成判断）。另外还会侦听植物卡片是否被选中。
定义结构体 GameMapInformation，对应植物种植地图。

6.2.5.6 GSZombiesAppearControl.h/.cpp 定义类 ZombiesAppearControl，控制僵尸出现波数。

6.2.5.7 GSButtonLayer.h/.cpp 定义类 GSButtonLayer 和 PlantsInformation，其中类 GSButtonLayer 用于设置对战界面的按钮，包括铲子按钮、加速减速游戏进程按钮、暂停按钮等；PlantsInformation 定义植物冷却时间和植物所需阳光。

6.2.5.8 GSGameResultJudgement.h/.cpp 定义类 GSGameResultJudgement，用于判断关卡是否结束，结束的条件根据关卡数据可能包含至少要收集多少阳光或保留多少小车。

6.2.5.9 GSGameEndLayer.h/.cpp 定义类 GSGameEndLayer。用于展示各类游戏结束界面，比如因不满足游戏要求或僵尸进到家里而失败，或满足要求通关成功（奖励金币）。

successfullEntry() 成功通关调用此方法，存储当前关卡数（可能废弃）、将存留下来的除草机转换为金币奖励、奖励一袋子金币。

breakThrough(GameTypes gameType) 根据游戏类型显示失败界面，如果是因为僵尸进到家里失败则显示僵尸吃掉了你的脑子；如果是因为不满足游戏要求失败（比如存留下来的除草机数量不满足要求），则显示通关要求对话框。

showFailText() 显示僵尸吃掉了你的脑子，并退出游戏。

showFailDialog(gameType) 显示通关要求对话框，强调失败原因。

rewardThing() 奖励一袋子金币。

6.2.6 文件夹 SelectPlantsScene

6.2.6.1 SelectPlantsScene.h/.cpp 定义类 SelectPlantsScene，选择植物界面，依次调用 SPSBackgroundLayer 和 SPSControlLayer 实现功能。

eventUpdate(float Time) 包含滚动回对战的草坪、选择植物、显示游戏开始文字三类事件，分别于每隔 0.22 秒、0.25 秒、0.3 秒刷新。为了调用对应事件，需要在 schedule 函数中指定不同的刷新时间。

createSelectPlantsDialog() 创建选择植物对话框（通过创建 SPSSpriteLayer 实现）的同时，每隔 0.22 秒调用 eventUpdate（即触发全图滚动预览事件）。

selectPlantsCallBack() 创建了 0.3 秒的定时器

replaceScene() 设置初始阳光数后跳转到 GameScene。

6.2.6.2 SPSBackgroundLayer.h/.cpp 定义类 SPSBackgroundLayer，创建选择植物界面的背景和僵尸类型的预览。

6.2.6.3 SPSControlLayer.h/.cpp 定义类 SPSControlLayer，创建退出按钮并显示用户名字，如果退出的话会调用 SPSQuitLayer 弹出退出对话层。

6.2.6.4 SPSRequriementLayer.h/.cpp 定义类 SPSRequriementLayer，通关要求对话层，具体细节通过 UserWinRequirement 实现，例如通关后要保留 5 辆小车，要求从游戏数据读取。

6.2.6.5 SPSSpriteLayer.h/.cpp 定义类 SPSSpriteLayer，创建选择植物界面的对话框，包括可以选择哪些植物。

7 Cocos2d-x 介绍

Cocos2d-x 是使用 C++ 开发的开源框架（又被称为游戏引擎），支持 C++/JavaScript/Lua 作为开发语言，同时允许跨平台部署。

像是刀塔传奇、梦幻西游、保卫萝卜都是使用 Cocos2d-x 开发的。

7.1 文献

https://docs.cocos.com/cocos2d-x/manual/en/#，官方文档，最好的入门学习资源，可选中文版本（内容与英文版有一定区别）。

https://github.com/chukong/programmers-guide-samples，英文版教程对应示例代码。

https://discuss.cocos2d-x.org/t/how-to-run-programmers-guide-samples/26447，示例代码在 Linux 上运行的方法，似乎不可行，建议在 Win 上测试。

https://docs.cocos2d-x.org/api-ref/，官方 API 文档，比较详细。

7.2 安装

7.2.1 Ubuntu 安装

以下仅介绍 Ubuntu 安装方式，其他平台参见 cocos2d-x 在 GitHub 的介绍，注意：官方教程的安装步骤已过时，最新步骤请参见 cocos2dx 的 Github 仓库。

1. 创建 cocos2dx 的 fork，进入 fork，修改文件.gitmodules 中的 git 协议为 https 协议，具体步骤参见 7.2.4。注意修改最新的 v4 branch 的同时，也要切换到 v3 branch 进行相应修改，因为实际开发使用 v3 branch。
2. 修改完成后，克隆 fork 仓库，此处项目地址为 https://github.com/Xi-Gong/cocos2d-x/tree/v3。也可以直接克隆好执行如下命令切换分支，但是不还没有验证切换的目标是原仓库还是克隆仓库。

   ```
   git checkout --track origin/v3
   ```

3. 按照 Git 官方网站给出的步骤安装依赖，此处执行 python2.7 download-deps.py，关于 python2.7 的安装和使用，参见 7.2.3。
4. 执行 git submodule update --init。
5. 创建一个新游戏，对于 Linux，需要先安装依赖，命令如下。其他步骤参见 7.2.5。

   ```
   cd cocos2d-x/build
   ./install-deps-linux.sh

   // 以下是安装的日志，有些依赖没有添加上，选择执行以下命令手动安装
   ./install-deps-linux.sh: 15: DEPENDS+= libxmu-dev: not found
   ./install-deps-linux.sh: 16: DEPENDS+= libglu1-mesa-dev: not found
   ./install-deps-linux.sh: 17: DEPENDS+= libgl2ps-dev: not found
   ./install-deps-linux.sh: 18: DEPENDS+= libxi-dev: not found
   ./install-deps-linux.sh: 19: DEPENDS+= libzip-dev: not found
   ./install-deps-linux.sh: 20: DEPENDS+= libpng-dev: not found
   ./install-deps-linux.sh: 21: DEPENDS+= libcurl4-gnutls-dev: not found
   ./install-deps-linux.sh: 22: DEPENDS+= libfontconfig1-dev: not found
   ./install-deps-linux.sh: 23: DEPENDS+= libsqlite3-dev: not found
   ./install-deps-linux.sh: 24: DEPENDS+= libglew-dev: not found
   ./install-deps-linux.sh: 25: DEPENDS+= libssl-dev: not found
   ./install-deps-linux.sh: 26: DEPENDS+= libgtk-3-dev: not found
   ```

```
./install-deps-linux.sh: 27: DEPENDS+= binutils: not found
./install-deps-linux.sh: 28: DEPENDS+= xorg-dev: not found
W: --force-yes is deprecated, use one of the options starting with --allow instead.

sudo apt install libxmu-dev libglu1-mesa-dev libgl2ps-dev libxi-dev libzip-dev libpng
```

6. 运行新游戏，如报错 undefined reference to '___powf_finite' (Ubuntu20.04 及以上可能出现此问题)，参照 7.2.2。

```
cd pvz/pvz
cocos run -p linux
```

7.2.2 undefined reference to '___powf_finite'

根本原因在于自 Ubuntu 20.04 开始移除了 fast-math 数学库。

https://stackoverflow.com/questions/63261220/link-errors-with-ffast-math-ffinite-math-only-and-glibc-2-31，通过自定义缺少的数学函数解决问题。

具体操作步骤为，进入/home/gx/Documents/GitHub/pvz/pvz/proj.linux，修改 main.cpp，增加如下代码。

```
#include <math.h>

extern "C" {
    double __exp_finite(double x) { return exp(x); }
    double __log_finite(double x) { return log(x); }
    double __pow_finite(double x, double y) { return pow(x, y); }

    float __expf_finite(float x) { return expf(x); }
    float __logf_finite(float x) { return logf(x); }
    float __powf_finite(float x, float y) { return powf(x, y); }
}
```

7.2.3 python2.7 的安装

https://stackoverflow.com/questions/61486624/python-is-python-2-package-something-new，通过创建符号链接的方式使用 python2.7。

https://askubuntu.com/questions/1239829/modulenotfounderror-no-module-named-distutils-util,python3 使用已经废弃的 distutils 包。

```
// python-is-python3 don't fit setup.py requirements
// sudo apt install python-is-python3 python3-distutils

sudo apt install python2.7

// 无需连接，直接调用python2.7即可
// sudo ln -s /bin/python2.7 /usr/bin/python
```

7.2.4 git clone 能访问 Github 但是 git submodule --init 就不能访问的问题

https://blog.csdn.net/yangjia_cheng/article/details/122179029，提出另外一种方法，删除子模块空文件夹下的.git 隐藏文件，再重新执行 submodule 更新，尚未尝试。

https://blog.csdn.net/qq_46695411/article/details/126273079，提出问题：git clone 能访问 Github 但是 git submodule --init 就不能访问，解决办法是替换.gitmodules 的路径为 fork 的仓库，但是效率不高。

https://blog.csdn.net/sunjindeng123/article/details/124246100，替换.gitmodules 的路径为 Github 镜像地址，尝试后发现不可行。

https://cloud.tencent.com/developer/article/1347791，介绍 git 协议。

更改 git 协议为 https 协议，推荐。

```
git clone https://github.com/cocos2d/cocos2d-x.git
```

注意：克隆时不要使用 recursive。

克隆后进入目标文件夹，我的路径是/home/gx/Documents/GitHub/cocos2d。

在该路径下打开隐藏文件.gitmodules，将其中 url 采用的协议从 git 更改为 https。

```
[submodule "tools/cocos2d-console"]
        path = tools/cocos2d-console
        url = https://github.com/cocos2d/cocos2d-console.git
[submodule "tools/bindings-generator"]
        path = tools/bindings-generator
        url = https://github.com/cocos2d/bindings-generator.git
[submodule "tests/cpp-tests/Resources/ccs-res"]
        path = tests/cpp-tests/Resources/ccs-res
        url = https://github.com/dumganhar/ccs-res.git
```

接下来执行同步子模块的 url，然后克隆。

```
git submodule sync
git submodule update --init
```

还有一种方法是手动下载子模块放在指定路径下，不推荐。

即使打开 VPN，执行 submodule 更新也失败了，观察到克隆的 submodule 网址和目标路径，准备手动下载 submodule，拷贝到目标路径下。

Submodule 'tests/cpp-tests/Resources/ccs-res' (git://github.com/dumganhar/ccs-res.git) registered for path 'tests/cpp-tests/Resources/ccs-res'

Submodule 'tools/bindings-generator' (git://github.com/cocos2d/bindings-generator.git) registered for path 'tools/bindings-generator'

Submodule 'tools/cocos2d-console' (git://github.com/cocos2d/cocos2d-console.git) registered for path 'tools/cocos2d-console'

以 ccs-res 为例，此处'github.com/dumganhar/ccs-res.git' 是其网址，目标路径是'tests/cpp-tests/Resources/ccs-res'。

综上，以 PowerShell 为例，拷贝以上三个子模块的命令如下，默认上述三个子模块已经下载好并解压到你的仓库路径下（yourRepoPath）。

```
Copy-Item <源文件夹> <新文件夹> -recurse -force

cd yourRepoPath/programmers-guide-samples/cpp/cocos2d

// create destination dir(may not be necessary)
mkdir tests/cpp-tests/Resources/ccs-res;
mkdir tools/bindings-generator;
mkdir tools/cocos2d-console;

// copy submodule to destination
Copy-Item ../../../ccs-res-3 tests/cpp-tests/Resources/ccs-res -recurse -force;
Copy-Item ../../../bindings-generator-3 tools/bindings-generator -recurse -force;
Copy-Item ../../../cocos2d-console-3 tools/cocos2d-console-3 -recurse -force;
```

7.2.5 创建新游戏

如有问题，通过 cocos -h 或 cocos —help 获取帮助。如要获取某一个具体选项的帮助，比如 new，则使用 cocos new -h 或 cocos new -help。

参照官方文档，编写以下创建 pvz 游戏的命令。

```
cd cocos2d-x
./setup.py
source FILE_TO_SAVE_SYSTEM_VARIABLE # source .bashrc
cocos new pvz -p com.gx.pvz -l cpp -d ../pvz
cd /home/gx/Documents/GitHub/pvz/pvz
```

7.2.6 官方教程示例代码配置

对于 Win10，依次执行如下命令。

```
$ cd programmers-guide-samples/cpp
$ git clone https://github.com/cocos2d/cocos2d-x.git cocos2d
$ cd cocos2d
$ git checkout v3.10    // programmers-guide-samples branch, not cocos2dx
$ git submodule update —init
$ download-deps.py
```

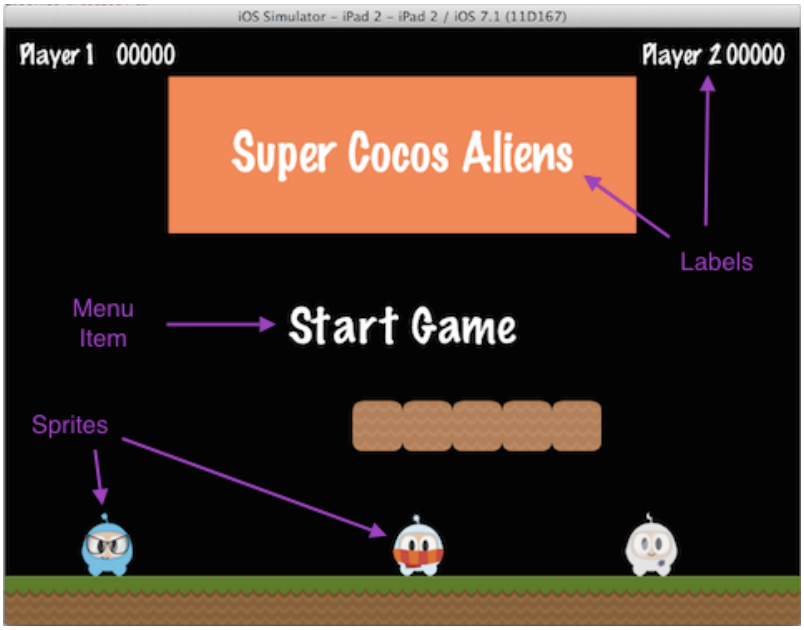
7.3 基本概念

cocos2d-x 通常包含下述组件：Renderer、2d/3d 图像、碰撞检测、物理引擎、声音、控制器支持、动画、序列等。

7.3.1 主要组件

Cocos2d-x 的核心包含 Scene, Node, Sprite, Menu, Menu 对象。

以下图游戏为例，可以发现若干组件。



对于植物大战游戏，可以发现如下组件。

其中植物可以看作精灵，阳光数量、击杀僵尸数量、金币数可以看作标签，暂停、关卡切换等可以看作菜单项。

7.3.2 搭建游戏

跨平台代码统一存放在 Classes 目录下，而平台相关代码分别存放在不同文件下，即 proj.android, proj.ios_mac, proj.linux, proj.win32。例子，https://github.com/ErLinErYi/PlantsVsZombies/tree/master/PlantsVsZombies。

AppDelegate 会在每个平台的代码运行时分别调用一次。

```
int main(int argc, char **argv)
{
    // create the application instance
    AppDelegate app;
    return Application::getInstance()->run();
}
```

设计的分辨率会影响精灵的大小的设计，以下是一些分辨率实例。

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

主要用于编码游戏的函数，AppDelegate::applicationDidFinishLaunching()。
https://blog.csdn.net/sunnyboychina/article/details/106124432，设计分辨率和屏幕分辨率的区别。

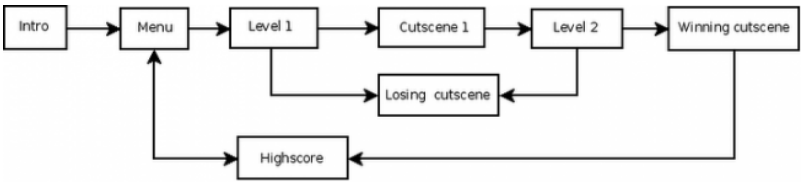
7.3.3 改变 glView 的标题

注意：这不会改变 IDE 项目的名称或是目录名称。

```
if(!glview) {
#if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN32) || (CC_TARGET_PLATFORM == CC_PLATFORM_MAC)
        glview = GLViewImpl::createWithRect("FirstGame", cocos2d::Rect(0, 0, designResol
#else
        glview = GLViewImpl::create("FirstGame");
#endif
        director->setOpenGLView(glview);
}
```

7.3.4 导演

把你自己想象成制片人，告诉导演（Director）应该怎么做。导演类一般用于场景切换，该类产生可共享的单一实例（singleton）。以下是游戏流程，导演在其中起到过渡的作用。



7.3.5 获取导演

```
// get the director and then use it
auto director = cocos2d::Director::getInstance();
director->runWithScene(scene);

// get the director for each operation (not recommended for repeated requests)
auto s = cocos2d::Director::getInstance()->getWinSize();
```

7.3.6 导演的行为

改变场景

```
director->runWithScene(scene); // use when starting your game

director->replaceScene(scene2); // use when changing from the running scene to another s
```

暂停和播放

```
// stop animations
cocos2d::Director::getInstance()->stopAnimation();

// resume animations
cocos2d::Director::getInstance()->startAnimation();
```

获取/设置内部属性

FPS（Frame Per Second）可以理解为刷新率，即画面每秒传输帧数，帧数越多越流畅。

需要指出，FPS 是由显卡的性能来决定，不是说显卡设置了刷新率 120HZ，帧率就会稳定输出 120HZ，它可能会波动变化，而刷新率 HZ 则是显示器的物理参数，它是固定不变的。

```
// turn on display FPS
cocos2d::Director::GetInstance()->setDisplayStats(true);

// set FPS. the default value is 1.0/60 if you don't call this
cocos2d::Director::GetInstance()->setAnimationInterval(1.0f / 60);

// set content scale factor
cocos2d::Director::GetInstance()->setContentScaleFactor(....);
```

Texture Cache

Texture Cache 是一个存储图片数据的只读 cache。

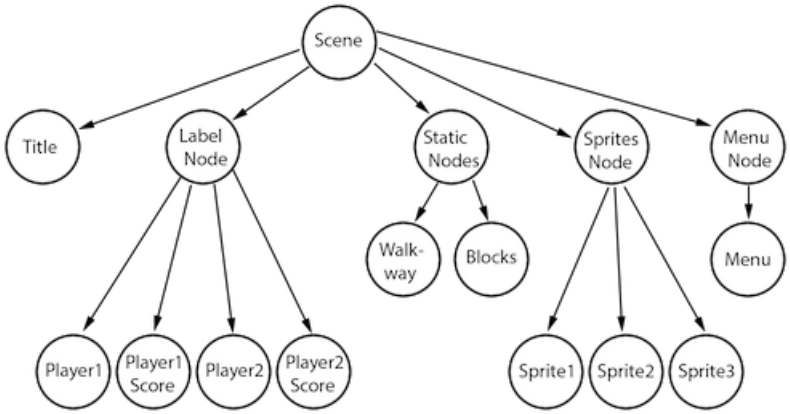
https://blog.csdn.net/wolf96/article/details/87884209，介绍 Texture Cache。

7.3.7 场景

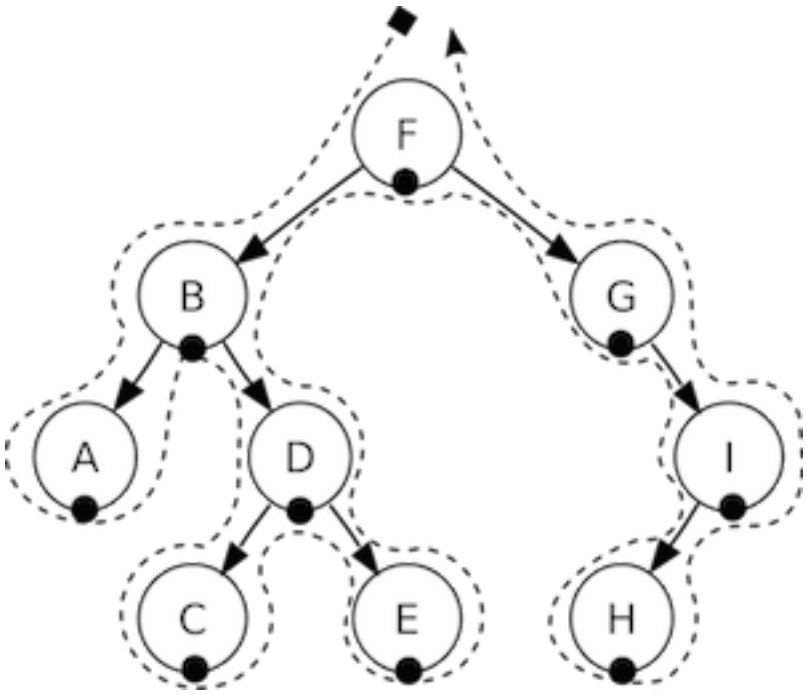
这是一个主菜单，由一个场景（Scene）构成，而该场景又由多个部分组成。场景由 renderer 绘制，renderer 负责组织所有出现在屏幕的部分，包括这些部分长什么样子，该如何出现。

7.3.8 场景图

场景图（Scene Graph）是一种安排场景内对象的数据结构，本质上是包含 Node 对象的树，上图对应的场景图如下。



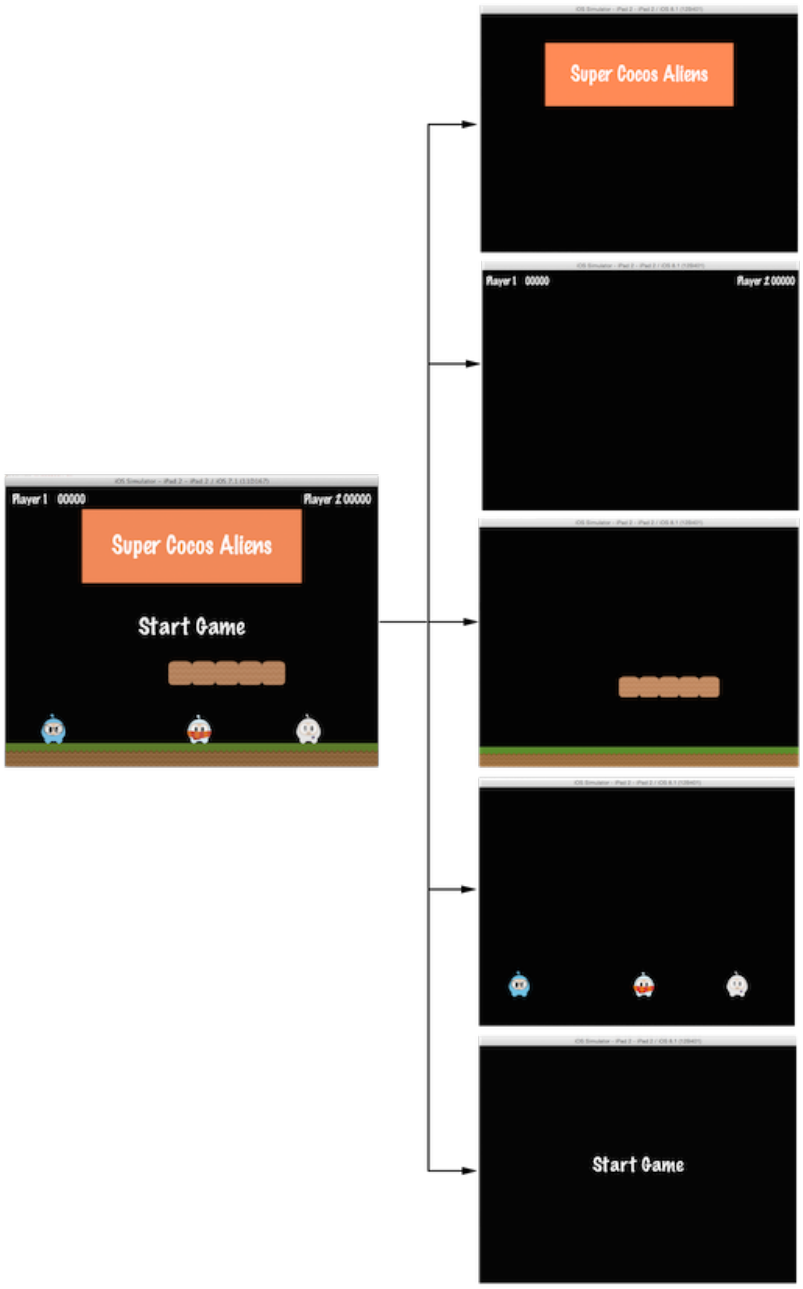
介绍场景图是为了解决开发中遇到对象无法正确显示的问题，比如精灵对象本来要显示在顶层，却进入了隐藏背景部分。cocos2d-x 会对场景图进行中序遍历（左根右），最后遍历（最右边）的对象将会显示在最顶层。



7.3.9 Z-Order

另一种理解的方式是定义 z-order，z-order 为负的元素在树的左侧，z-order 为正的元素在树的右侧。换句话说，z-order 为正的元素相比为负的元素显示在顶层。你可以添加任意次序的任意元素，这些元素会根据自定义的 z-order 排序。

如下图所示，这些 Node 对象按照一定的 z-order 堆叠起来。

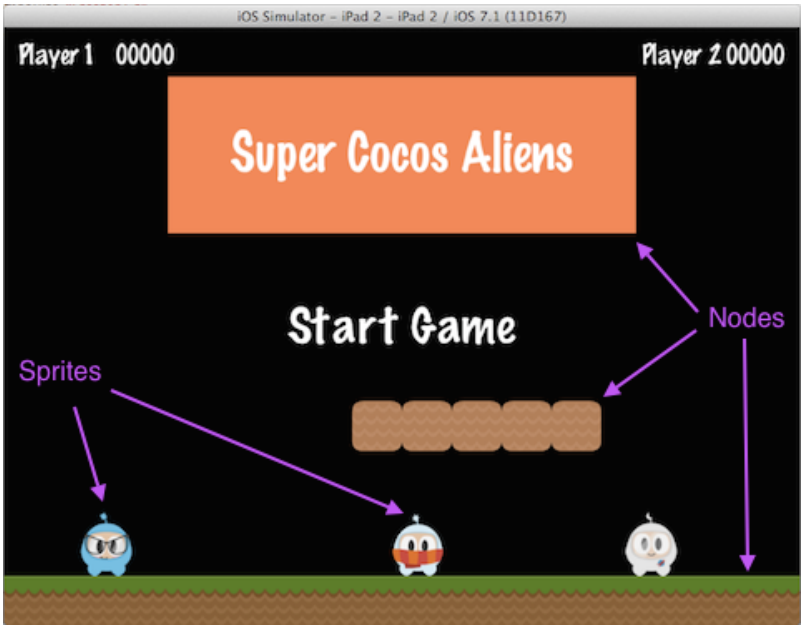


在 cocos2d-x 中使用 addChild() 建立场景图。

```
// Adds a child with the z-order of -2, that means
// it goes to the "left" side of the tree (because it is negative)
scene->addChild(title_node, -2);

// When you don't specify the z-order, it will use 0
scene->addChild(label_node);

// Adds a child with the z-order of 1, that means
// it goes to the "right" side of the tree (because it is positive)
scene->addChild(sprite_node, 1);
```

7.3.10 精灵

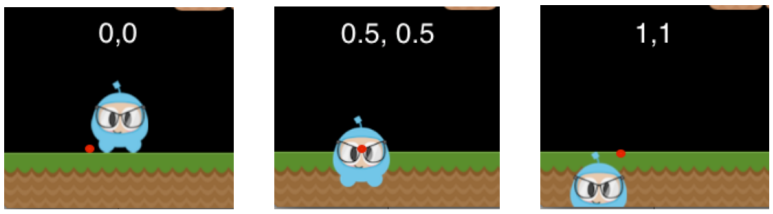
每个游戏都有这样的情景：一个舞台，上面站着一个某种形式的主角，那主角就是精灵（Sprite）。精灵是可以移动的物体，不能移动的物体只是 Node，如下图所示。



精灵拥有一些可以被配置的属性，比如：位置，旋转角度，缩放比例，透明度，颜色等等。

```
// This is how to create a sprite
auto mySprite = Sprite::create("mysprite.png");

// this is how to change the properties of the sprite
mySprite->setPosition(Vec2(500, 0));

mySprite->setRotation(40);

mySprite->setScale(2.0); // sets both the scale of the X and Y axis uniformly

mySprite->setAnchorPoint(Vec2(0, 0));
```

值得一提的是锚点（anchor point）属性，该属性是 setPosition() 的基础。实际上，setPosition() 是设置锚点的位置，而锚点的位置由 setAnchorPoint() 设置。所有 Node 对象都拥有一个锚点值（Sprite 是 Node 的子类）。



7.3.11 动作

动作（Action）允许 Node 对象在时域上进行转换，你可以在 Node 对象上创建一个动作序列，动作包括移动位置（Move by/Move to）、旋转（Rotate）、伸缩（Scale）等。

移动精灵的示例代码如下，其中 MoveBy 执行两次移动，而 MoveTo 直接向目标位置移动。

```
auto mySprite = Sprite::create("Blue_Front1.png");

// Move a sprite 50 pixels to the right, and 10 pixels to the top over 2 seconds.
auto moveBy = MoveBy::create(2, Vec2(50,10));
mySprite->runAction(moveBy);

// Move a sprite to a specific location over 2 seconds.
auto moveTo = MoveTo::create(2, Vec2(50,10));
mySprite->runAction(moveTo);
```

7.3.12 序列和并发

序列（Sequence）安排多个动作对象按一定顺序执行。

图示序列对应代码如下。


```
auto mySprite = Node::create();

// move to point 50,10 over 2 seconds
auto moveTo1 = MoveTo::create(2, Vec2(50,10));

// move from current position by 100,10 over 2 seconds
auto moveBy1 = MoveBy::create(2, Vec2(100,10));

// move to point 150,10 over 2 seconds
auto moveTo2 = MoveTo::create(2, Vec2(150,10));

// create a delay
auto delay = DelayTime::create(1);

mySprite->runAction(Sequence::create(moveTo1, delay, moveBy1, delay.clone(),
moveTo2, nullptr));
```

另外，Spawn 类可以使所有动作同时开始执行（即并发），但是各动作不一定同时结束（取决于动作的持续时间）。并发的场景可以是击败 Boss 时需要同时使用多个攻击手段。

```
auto myNode = Node::create();

auto moveTo1 = MoveTo::create(2, Vec2(50,10));
```

```
auto moveBy1 = MoveBy::create(2, Vec2(100,10));
auto moveTo2 = MoveTo::create(2, Vec2(150,10));

myNode->runAction(Spawn::create(moveTo1, moveBy1, moveTo2, nullptr));
```

7.3.13 结点关系

cocos2d-x 的结点（Node）存在父子关系，更改父结点会同时应用于其下的子节点。
以下代码旋转父结点，子节点也一同旋转。


```
auto myNode = Node::create();

// rotating by setting
myNode->setRotation(50);
```

需要注意的是，不是所有的父节点属性都会被自动应用到子节点，如改变父节点的锚点只会影响转换效果 (比例缩放，位置变化，角度旋转，变形等)，不会影响子节点锚点，子节点的锚点总会是左下角 (0,0)。

7.3.14 日志

比起 C++ 的 std::cout，log 使用格式化字符串输出更方便，类似于 fmt 库。
注意输出 std::string 时，需要通过 c_str() 函数转换为 const char *，才能正常输出，否则会乱码。
另外只有在 Debug 模式下，才会输出日志信息。

```
// a simple string
log("This would be outputted to the console");

// a string and a variable
string s = "My variable";
cocos2d::log("string is %s", s.c_str());
```

7.4 精灵

精灵（Sprites）实际上就是一张 2D 图片，可以制作成动画或是改变其属性实现转换，比如旋转，移动位置，缩放等。

7.4.1 创建精灵

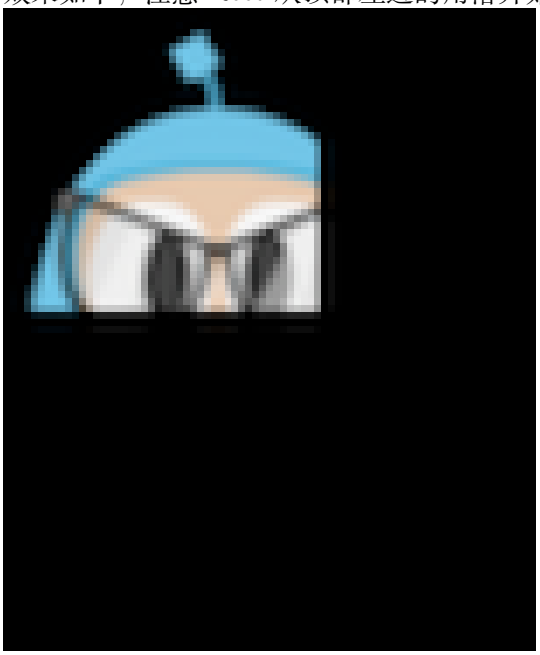
可以从多种图片格式（如 PNG，JPEG）创建精灵。创建精灵时默认使用图片本身的大小，即图片若是 200*200，则精灵的大小也是 200*200。

```
auto mySprite = Sprite::create("mysprite.png");
```

创建精灵时也可以指定使用图片的一部分，此处使用 Rect 划定，四个参数依次为原点 x、原点 y、宽、高。

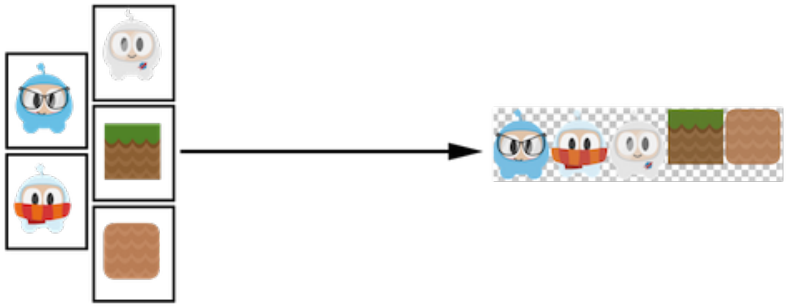
```
auto mySprite = Sprite::create("mysprite.png", Rect(0,0,40,40));
```

效果如下，注意 Rect 从顶部左边的角落开始计算。



7.4.2 精灵图集

精灵图集就是将精灵们合并到一个文件中，通过图集可以实现批量调用绘制函数，提高效率。



7.4.3 加载精灵图集

使用精灵图集时，第一次要全部加载到 SpriteFrameCache 实例中。类 SpriteFrameCache 是一个缓存类，会保留添加到其中的 SpriteFrame 对象（元图片）。如此一来，SpriteFrame 对象只需加载一次，并保留在 SpriteFrameCache 中。

```
// load the Sprite Sheet
auto spritecache = SpriteFrameCache::getInstance();

// the .plist file can be generated with any of the tools mentioned below
spritecache->addSpriteFramesWithFile("sprites.plist");
```

7.4.4 制作精灵图集

推荐使用 Texture Packer，有 linux 版本，https://www.codeandweb.com/texturepacker。

7.4.5 SpriteFrame Cache

7.4.6 从 SpriteFrame 名称创建

```
// Our .plist file has names for each of the sprites in it.  We'll grab
// the sprite named, "mysprite" from the sprite sheet:
auto mysprite = Sprite::createWithSpriteFrameName("mysprite.png");
```

7.4.7 从 SpriteFrame 创建

```
// this is equivalent to the previous example,
// but it is created by retrieving the SpriteFrame from the cache.
auto newspriteFrame = SpriteFrameCache::getInstance()->getSpriteFrameByName("Blue_Front1
auto newSprite = Sprite::createWithSpriteFrame(newspriteFrame);
```

7.4.8 精灵操纵

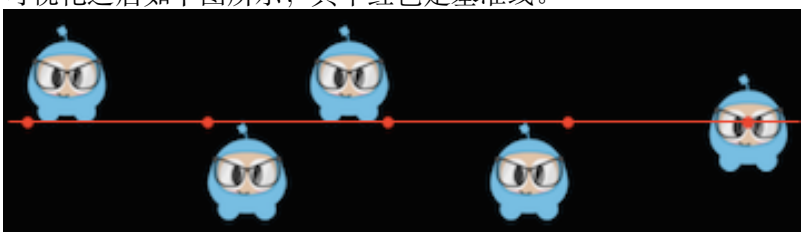
7.4.9 锚点

锚点确定了精灵对象在计算坐标位置的一个基准点，这个点是精灵内部的点，锚点影响精灵的缩放，旋转，倾斜这种转换，不影响颜色，透明度这种属性。锚点使用的坐标系以左下角为原点 (0, 0)，默认情况下，所有 Node 对象的锚点是 (0.5, 0.5)。

```
// DEFAULT anchor point for all Sprites
mySprite->setAnchorPoint(0.5, 0.5);

// bottom left
mySprite->setAnchorPoint(0, 0);

// top left
mySprite->setAnchorPoint(0, 1);
```

```
// bottom right
mySprite->setAnchorPoint(1, 0);

// top right
mySprite->setAnchorPoint(1, 1);
```

可视化之后如下图所示，其中红色是基准线。



7.4.10 位置

精灵的位置通过锚点作为基准点来确定，当我们修改锚点值时，精灵的位置也会随之改变。

7.4.11 旋转

以锚点作为基准点旋转。精灵旋转的角度值为正或负，正值会顺时针旋转精灵对象，负值会逆时针旋转，默认旋转角度为 0。

```
// rotate sprite by +20 degrees
mySprite->setRotation(20.0f);

// rotate sprite by -20 degrees
mySprite->setRotation(-20.0f);

// rotate sprite by +60 degrees
mySprite->setRotation(60.0f);

// rotate sprite by -60 degrees
mySprite->setRotation(-60.0f);
```

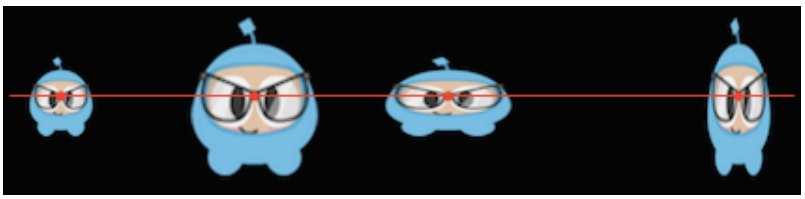
可视化之后如下图所示，其中红色是基准线。

7.4.12 缩放

精灵可以沿着 x 轴、y 轴或 xy 轴同时缩放。默认的缩放值是 1，沿 xy 轴缩放。

```
// increases X and Y size by 2.0 uniformly
mySprite->setScale(2.0);
```

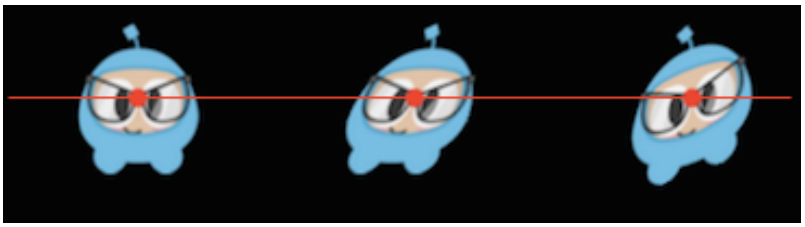
```
// increases just X scale by 2.0
mySprite->setScaleX(2.0);

// increases just Y scale by 2.0
mySprite->setScaleY(2.0);
```



7.4.13 倾斜

和缩放相同，精灵可以沿 x 轴、y 轴或 xy 轴同时倾斜。默认的倾斜值是 1，沿 xy 轴倾斜。

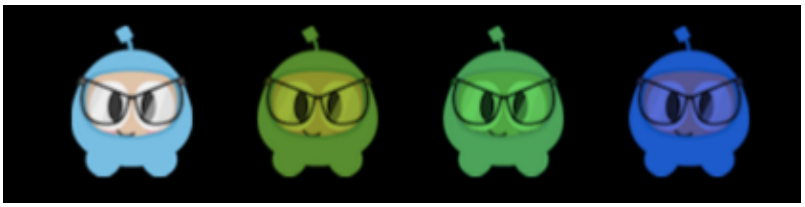
```
// adjusts the X skew by 20.0
mySprite->setSkewX(20.0f);

// adjusts the Y skew by 20.0
mySprite->setSkewY(20.0f);
```



7.4.14 颜色

通过给出包含 RGB 值的 Color3B 对象，设置精灵的颜色。cocos2dx 还提供了预定义的颜色，如 Color3B::Red。

```
// set the color by passing in a pre-defined Color3B object.
mySprite->setColor(Color3B::WHITE);

// Set the color by passing in a Color3B object.
mySprite->setColor(Color3B(255, 255, 255)); // Same as Color3B::WHITE
```



7.4.15 透明度

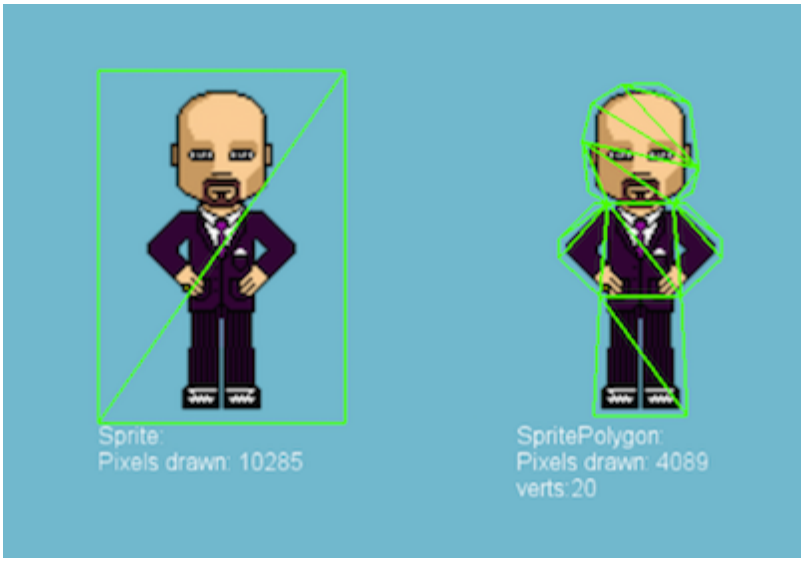
透明度范围在 0-255，255 表示完全不透明，0 表示完全透明（不可见）。默认透明度为 255。

```
// Set the opacity to 30, which makes this sprite 11.7% opaque.
// (30 divided by 256 equals 0.1171875...)
mySprite->setOpacity(30);
```

7.4.16 多边形精灵

为了提高 GPU 处理图片的效率，使用多边形精灵的概念。

原本按照像素绘制，一个精灵仅用两个三角形就可以绘制；现在为了节省绘制资源，改为沿顶点绘制，绘制时划分为更多三角形。




```
// Generate polygon info automatically.
auto pinfo = AutoPolygon::generatePolygon("filename.png");

// Create a sprite with polygon info.
auto sprite = Sprite::create(pinfo);
```

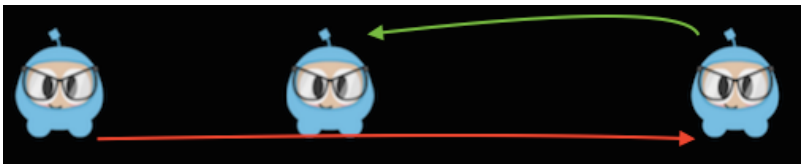
7.5 动作

动作（Action）对象通过修改结点属性制造变换。

7.5.1 MoveBy 和 MoveTo 的区别

MoveBy 是在当前位置的基础上移动，是相对的。MoveTo 是直接移动到给定位置，是绝对的。

```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setPosition(Vec2(200, 256));

// MoveBy - lets move the sprite by 500 on the x axis over 2 seconds
// MoveBy is relative - since x = 200 + 500 move = x is now 700 after the move
auto moveBy = MoveBy::create(2, Vec2(500, mySprite->getPositionY()));

// MoveTo - lets move the new sprite to 300 x 256 over 2 seconds
// MoveTo is absolute - The sprite gets moved to 300 x 256 regardless of
// where it is located now.
auto moveTo = MoveTo::create(2, Vec2(300, mySprite->getPositionY()));

// Delay - create a small delay
auto delay = DelayTime::create(1);

auto seq = Sequence::create(moveBy, delay, moveTo, nullptr);

mySprite->runAction(seq);
```



7.5.2 基本动作

7.5.3 Move, Rotate, Scale, Tint, Fade

这些基本动作都有共通的用法，每个动作都有 XXBy 和 XXTo 方法，分别对应相对和绝对的操作。Fade 略有区别，对应 Fade In/Out。

7.5.4 Animate

Animate 对象很容易实现“翻页式”的动画，即每隔一个短暂时间进行图像替代。

```
auto mySprite = Sprite::create("mysprite.png");

// now lets animate the sprite we moved
Vector<SpriteFrame*> animFrames;
animFrames.reserve(12);
animFrames.pushBack(SpriteFrame::create("Blue_Front1.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Front2.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Front3.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Left1.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Left2.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Left3.png", Rect(0,0,65,81)));
```

```
animFrames.pushBack(SpriteFrame::create("Blue_Back1.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Back2.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Back3.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Right1.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Right2.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("Blue_Right3.png", Rect(0,0,65,81)));

// create the animation out of the frames
Animation* animation = Animation::createWithSpriteFrames(animFrames, 0.1f);
Animate* animate = Animate::create(animation);

// run it and repeat it forever
mySprite->runAction(RepeatForever::create(animate));
```

7.5.5 Easing（变速运动）

变速动作可以让节点对象具有加速度，产生平滑同时相对复杂的动作，所以可以用变速动作来模仿一些物理运动，这样比实际使用物理引擎的性能消耗低，使用起来也简单。

```
// create a sprite
auto mySprite = Sprite::create("mysprite.png");

// create a MoveBy Action to where we want the sprite to drop from.
auto move = MoveBy::create(2, Vec2(200, dirs->getVisibleSize().height -
 newSprite2->getContentSize().height));

// create a BounceIn Ease Action
auto move_ease_in = EaseBounceIn::create(move->clone() );
auto move_ease_in_back = move_ease_in->reverse();

// create a delay that is run in between sequence events
auto delay = DelayTime::create(0.25f);

// create the sequence of actions, in the order we want to run them
auto seq1 = Sequence::create(move_ease_in, delay, move_ease_in_back,
    delay->clone(), nullptr);

// run the sequence and repeat forever.
mySprite->runAction(RepeatForever::create(seq1));
```

7.5.6 序列

7.5.7 加入函数

序列（Sequence）可以连续执行的一系列 Action 对象。除了 Action 对象之外，为了增加灵活性，序列中还可以包含函数，即 CallFunc 对象。

CallFunc::create 可以创建回调函数，回调函数的定义：如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用为调用它所指向的函数时，我们就说这是回调函数。

```
auto mySprite = Sprite::create("mysprite.png");

// create a few actions.
auto jump = JumpBy::create(0.5, Vec2(0, 0), 100, 1);

auto rotate = RotateTo::create(2.0f, 10);

// create a few callbacks
auto callbackJump = CallFunc::create([](){
    log("Jumped!");
});

auto callbackRotate = CallFunc::create([](){
    log("Rotated!");
});

// create a sequence with the actions and callbacks
auto seq = Sequence::create(jump, callbackJump, rotate, callbackRotate, nullptr);

// run it
mySprite->runAction(seq);
```

7.5.8 加入延迟

```
cocos2d::DelayTime* delay = cocos2d::DelayTime::create(1);

// A, B, C is callback fun created by CallFunc
runAction(cocos2d::Sequence::create(A, delay, B, delay, C, NULL));
```

7.5.9 Spawn

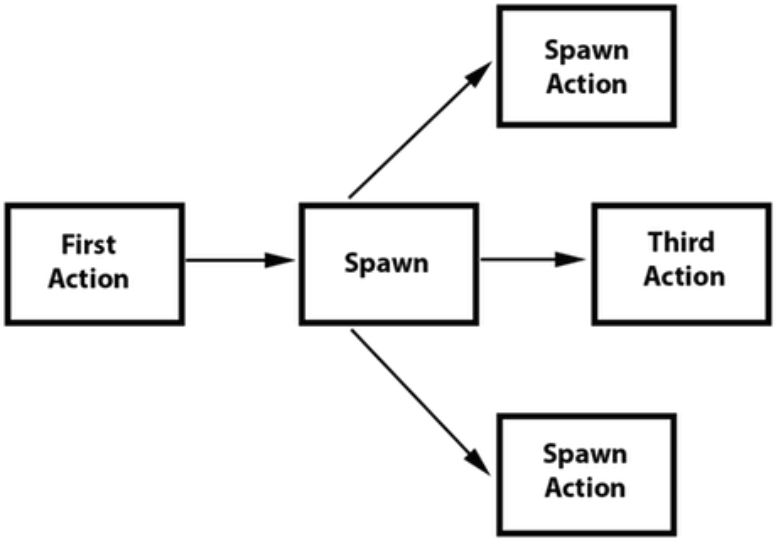
如前所述，Spawn 对象允许同时执行多个 Action 对象，其效果和连续执行相同。

使用 Spawn 对象可以获得连续执行所不能获得的效果。

```
// create 2 actions and run a Spawn on a Sprite
auto mySprite = Sprite::create("mysprite.png");
```

```
auto moveBy = MoveBy::create(10, Vec2(400,100));
auto fadeTo = FadeTo::create(2.0f, 120.0f);

// running the above Actions with Spawn.
auto mySpawn = Spawn::createWithTwoActions(moveBy, fadeTo);
mySprite->runAction(mySpawn);
```

另外可以在序列中包含 Spawn 对象，如下图所示。




```
// create a Sprite
auto mySprite = Sprite::create("mysprite.png");

// create a few Actions
auto moveBy = MoveBy::create(10, Vec2(400,100));
auto fadeTo = FadeTo::create(2.0f, 120.0f);
auto scaleBy = ScaleBy::create(2.0f, 3.0f);

// create a Spawn to use
auto mySpawn = Spawn::createWithTwoActions(scaleBy, fadeTo);

// tie everything together in a sequence
auto seq = Sequence::create(moveBy, mySpawn, moveBy, nullptr);

// run it
mySprite->runAction(seq);
```

7.5.10 Clone 和 Reverse

7.5.11 Clone

当我们将同一动作施加到多个 Node 对象时，需要考虑 Action 对象的拷贝问题。原因在于每个 Action 对象都有自己的内部状态（internal state），Action 对象执行之后，其内部状态也会改变，如果再次使用同一对

象会出现难以预料的结果。

考虑下面的例子。

如果有一个 heroSprite 对象在原点 (0, 0) 处，在其上执行以下 Action 对象。

```
MoveBy::create(10, Vec2(400,100));
```

执行后，heroSprite 会被移动到 (400, 100) 处，此时该 Action 对象的内部状态也发生了改变，其内记录了目前的位置是 (400, 100)。

现在引入 enemySprite 对象，在 (200, 200) 处，对其施加相同的 Action 对象，最终 enemySprite 会被移动到 (800, 200) 处（相对 Action 对象的内部状态），而非预期的位置。

而 Clone 方法可以解决此问题，该方法返回引用（Ref 对象）的拷贝，是深拷贝。

```
// create our Sprites
auto heroSprite = Sprite::create("herosprite.png");
auto enemySprite = Sprite::create("enemysprite.png");

// create an Action
auto moveBy = MoveBy::create(10, Vec2(400,100));

// run it on our hero
heroSprite->runAction(moveBy);

// run it on our enemy
enemySprite->runAction(moveBy->clone()); // correct! This will be unique
```

7.5.12 Reverse

调用 reverse() 方法不仅仅是反向执行序列或 Spawn 中的动作，实际上是以相反的方向操纵序列或 Spawn 中的属性。

```
// reverse a sequence, spawn or action
mySprite->runAction(mySpawn->reverse());
```

7.6 场景

场景（Scene）中包含创建游戏所需的 Sprites, Labels, Nodes 对象等。游戏内可以有许多 Scene 对象并在之间切换，切换效果也可以自定义。

7.6.1 创建场景

创建场景很简单。

```
auto myScene = Scene::create();
```

7.6.2 scene(), create(), init() 调用关系

https://blog.csdn.net/wozhengtao/article/details/54600199。

示例。

```
class LoadingScene :public Scene
{
public:
        // 生成LoadingScene的create函数
        CREATE_FUNC(LoadingScene);
        // ...
};

/** @def CREATE_FUNC(__TYPE__)
 * Define a create function for a specific type, such as Layer.
 *
 * @param __TYPE__  class type to add create(), such as Layer.
 */
#define CREATE_FUNC(__TYPE__) \
static __TYPE__* create() \
{ \
    __TYPE__ *pRet = new(std::nothrow) __TYPE__(); \
        // create函数会调用所给类中的init成员函数
    if (pRet && pRet->init()) \
    { \
        pRet->autorelease(); \
        return pRet; \
    } \
    else \
    { \
        delete pRet; \
        pRet = nullptr; \
        return nullptr; \
    } \
}
```

7.6.3 场景图和 Z-Order

场景图决定了场景内节点对象的渲染顺序，也要记得 z-order 是如何影响场景图的。

7.6.4 一个简单场景

Cocos2d-x 用右手坐标系，也就是说坐标原点 (0,0) 在展示区的左下角，当你在场景里放置一些节点对象设置坐标位置时，注意左下角是坐标计算的起点。

```
auto dirs = Director::getInstance();
Size visibleSize = dirs->getVisibleSize();

auto myScene = Scene::create();
```

```
auto label1 = Label::createWithTTF("My Game", "Marker Felt.ttf", 36);
label1->setPosition(Vec2(visibleSize.width / 2, visibleSize.height / 2));

myScene->addChild(label1);

auto sprite1 = Sprite::create("mysprite.png");
sprite1->setPosition(Vec2(100, 100));

myScene->addChild(sprite1);
```

7.6.5 场景切换

7.6.6 切换方法

场景切换有很多方式，每种方式对应的方法通过如下模式调用。

```
Director::getInstance()->XXScene(myScene);
```

runWithScene()，用于开始游戏，加载第一个场景（只用于第一个场景!）。

replaceScene()，使用传入的场景替换当前场景来切换画面，当前场景被释放。这是切换场景时最常用的方法。

pushScene()，将当前运行中的场景暂停并压入到场景栈中，再将传入的场景设置为当前运行场景。只有存在正在运行的场景时才能调用该方法。

popScene()，释放当前场景，再从场景栈中弹出栈顶的场景，并将其设置为当前运行场景。如果栈为空，直接结束应用。

7.6.7 切换效果

```
auto myScene = Scene::create();

// Transition Fade
Director::getInstance()->replaceScene(TransitionFade::create(0.5, myScene, Color3B(0,255

// FlipX
Director::getInstance()->replaceScene(TransitionFlipX::create(2, myScene));

// Transition Slide In
Director::getInstance()->replaceScene(TransitionSlideInT::create(1, myScene) );
```

7.7 UI 组件

像是 labels, buttons, menus, sliders and views 都是 UI 组件。

7.7.1 标签

Cocos2d-x 提供标签（Label）对象给用户，可以使用位图字体，TrueType 字体，系统字体创建标签。

7.7.2 位图字体

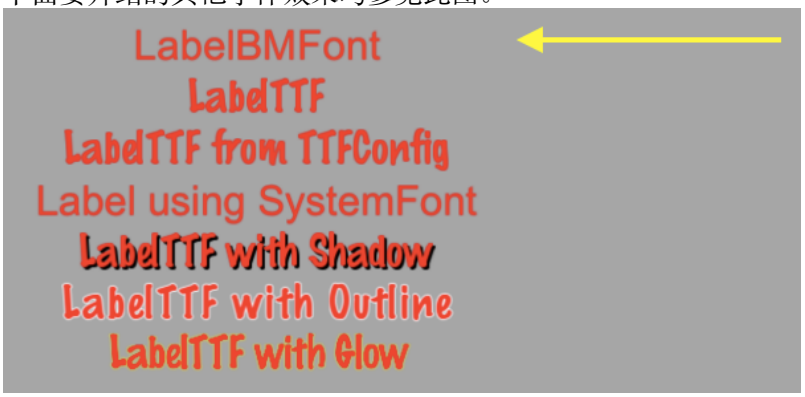
位图字体（BMFont）是一个使用位图字体创建的标签类型，位图字体中的字符由点阵组成。这种字体标签性能好，但是缩放会失真。标签中的每个字符都是一个单独的 Sprite，也就是说精灵的属性控制适用于这里的每个字符。

创建 BMFont 标签需要两个文件：.fnt 文件和.png 文件。

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

注意：在标签中出现的字符应能在提供的.fnt 文件找到，如果找不到，字符就不会被渲染。

下面要介绍的其他字体效果均参见此图。



7.7.3 TrueType 字体

TrueType 字体（TTF）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。

虽然使用 TrueType 字体更灵活，但是显示的时间更长，并且更改如 font face 和 size 的属性需要付出昂贵的代价。

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);
```

7.7.4 TTF Config

如果你创建了多个相同属性的 TrueType 字体，那么可以使用 TTFConfig 管理他们。TTFConfig 允许你设置 TTF 对象的共有属性。

另外 TTFConfig 还可以用于显示中文。

```
// 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");
```

7.7.5 系统字体

系统字体（System Font）是使用默认系统字体和字体大小的 Label 对象，因此其属性不可改变。

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

7.7.6 字体特效

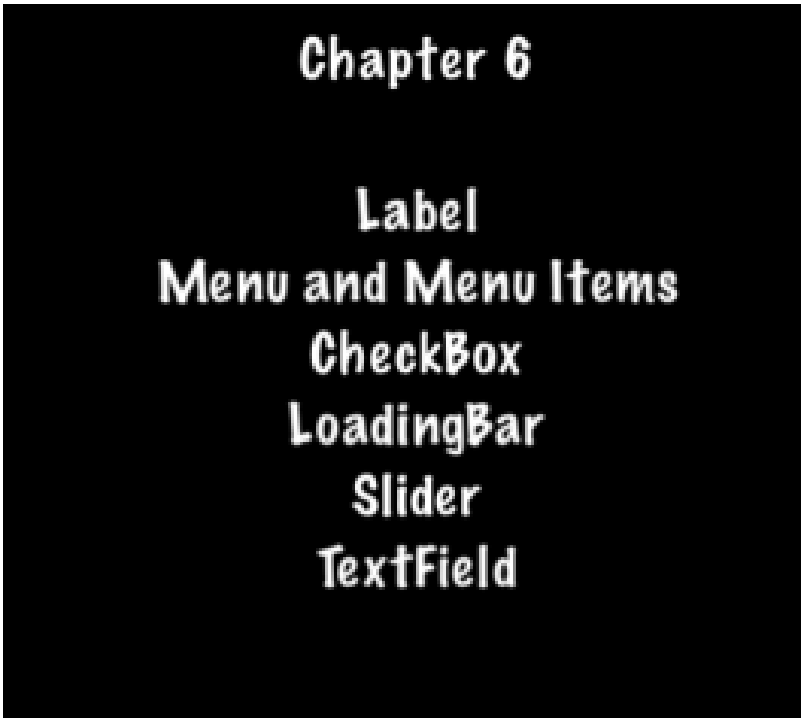
Cocos2dx 提供字体特效，覆盖大部分需求，因此无需自定义字体。不同字体支持特定的特效。

```
auto myLabel = Label::createWithTTF("My Label Text", "myFont.ttf", 16);

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

// 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);
```

7.7.7 菜单

菜单通常包含开始、退出、设置、关于等项，也可以包含子菜单，没有菜单项的菜单没有存在的意义。Cocos2d-x 提供 Menu 对象支持菜单功能，Menu 对象是一种特殊的 Node 对象。

菜单项一般有正常状态和选择状态。菜单项显示时是正常状态，当你点击它时变为选择状态，同时点击菜单还会触发一个回调函数。



7.7.8 创建菜单

以下给出创建菜单的一些方法，包括创建空菜单，使用图像创建菜单，使用包含 MenuItem 的 vector 创建菜单。

```
// creating a emtpy menu
auto myMenu = Menu::create();

// -------------------------------------
// creating a menu with a single item

// create a menu item by specifying images
auto closeItem = MenuItemImage::create("CloseNormal.png", "CloseSelected.png",
CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

auto menu = Menu::create(closeItem, NULL);
this->addChild(menu, 1);

// -------------------------------------
// creating a Menu from a Vector of items
Vector<MenuItem*> MenuItems;

auto closeItem = MenuItemImage::create("CloseNormal.png", "CloseSelected.png",
CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

MenuItems.pushBack(closeItem);
```

```
/* repeat for as many menu items as needed */

auto menu = Menu::createWithArray(MenuItems);
this->addChild(menu, 1);
```

7.7.9 使用 Lambda 表达式

点击菜单项将会触发回调函数，该函数可以是 Lambda 表达式。

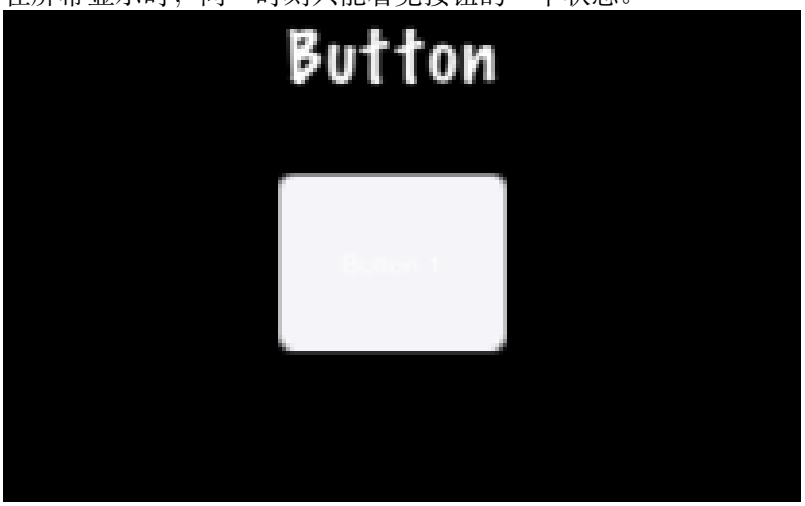
注意：Lambda 表达式在运行时（runtime）检查，而非编译时。

```
auto closeItem = MenuItemImage::create("CloseNormal.png", "CloseSelected.png",
[&](Ref* sender){
    // your code here
});
```

7.7.10 按钮

按钮会拦截点击事件，事件触发时调用事先定义好的回调函数。按钮有一个正常状态，一个选择状态，还有一个不可点击状态，按钮的外观可以根据这三个状态而改变。记得在操作的时候要包含头文件：#include "ui/CocosGUI.h"。

在屏幕显示时，同一时刻只能看见按钮的一个状态。



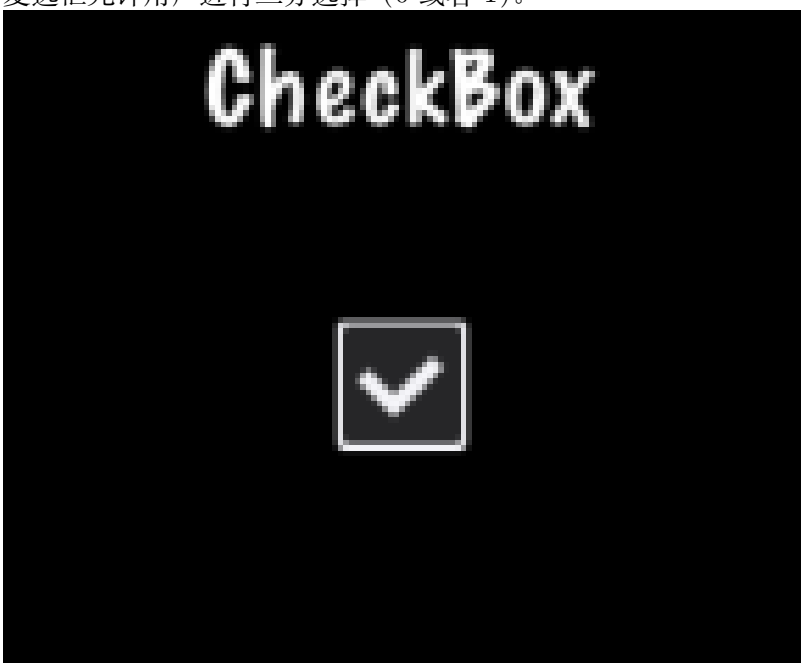
以下代码为按钮每个状态都指定了一个 png 图像。

```
auto button = Button::create("normal_image.png", "selected_image.png", "disabled_image.p

button->setTitleText("Button Text");
```

```
button->addTouchEventListener([&](Ref* sender, Widget::TouchEventType type){
        switch (type)
        {
                case ui::Widget::TouchEventType::BEGAN:
                        break;
                case ui::Widget::TouchEventType::ENDED:
                        std::cout << "Button 1 clicked" << std::endl;
                        break;
                default:
                        break;
        }
});

this->addChild(button);
```

7.7.11 复选框

复选框允许用户进行二分选择 (0 或者 1)。



复选框有 5 种状态: 正常状态、正常按下状态、选择状态、正常不可点击状态、选择不可点击状态, 每种状态对应一张图片。

```
#include "ui/CocosGUI.h"

auto checkbox = CheckBox::create("check_box_normal.png",
```

```
                                        "check_box_normal_press.png",
                                        "check_box_active.png",
                                        "check_box_normal_disable.png",
                                        "check_box_active_disable.png");

checkbox->addTouchEventListener([&](Ref* sender, Widget::TouchEventType type){
        switch (type)
        {
                case ui::Widget::TouchEventType::BEGAN:
                        break;
                case ui::Widget::TouchEventType::ENDED:
                        std::cout << "checkbox 1 clicked" << std::endl;
                        break;
                default:
                        break;
        }
});

this->addChild(checkbox);
```

7.7.12 进度条

游戏加载一般都需要进度条。

进度条的图片。

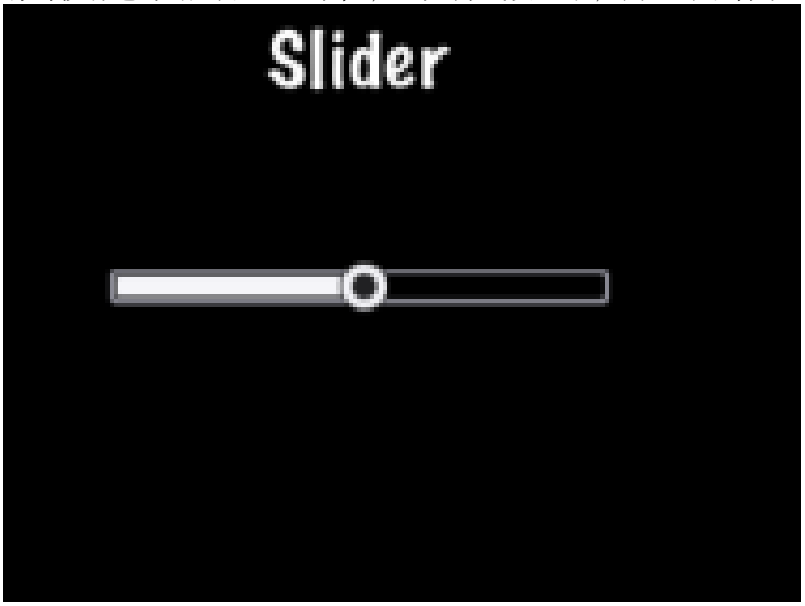
以下代码创建了一个进度条，设置了当进度增加时，进度条向右填充，在一些事件发生后同时改变进度条的进度。

```
#include "ui/CocosGUI.h"

auto loadingBar = LoadingBar::create("LoadingBarFile.png");
loadingBar->setDirection(LoadingBar::Direction::RIGHT);
```

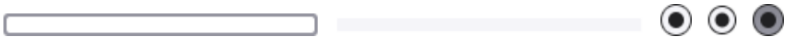
```
// something happened, change the percentage of the loading bar
loadingBar->setPercent(25);

// more things happened, change the percentage again.
loadingBar->setPercent(35);

this->addChild(loadingBar);
```

7.7.13 滑动条

有时候你想平滑的改变一个值，比如游戏设置中，调整背景音乐的音量，这时就要用到滑动条（Slider）。



实现一个滑动条需要提供五张图像，对应滑动条的不同部分不同状态，分别为：滑动条背景，上层进度条，正常显示时的滑动端点，滑动时的滑动端点，不可用时的滑动端点。



```
#include "ui/CocosGUI.h"

auto slider = Slider::create();
slider->loadBarTexture("Slider_Back.png"); // what the slider looks like
slider->loadSlidBallTextures("SliderNode_Normal.png", "SliderNode_Press.png", "SliderNod
slider->loadProgressBarTexture("Slider_PressBar.png");

slider->addTouchEventListener([&](Ref* sender, Widget::TouchEventType type){
        switch (type)
        {
                case ui::Widget::TouchEventType::BEGAN:
                        break;
                case ui::Widget::TouchEventType::ENDED:
```

```
                        std::cout << "slider moved" << std::endl;
                        break;
                default:
                        break;
        }
});

this->addChild(slider);
```

7.7.14 文本框

Cocos2d-x 提供 TextField 满足输入文本的需求。它支持触摸事件，焦点，定位内容百分比等。提供的文本框对象，是多功能的，能满足所有的输入需求，比如用户密码的输入，限制用户可以输入的字符数等等!

```
#include "ui/CocosGUI.h"

auto textField = TextField::create("","Arial",30);

// make this TextField password enabled
textField->setPasswordEnabled(true);

// set the maximum number of characters the user can enter for this TextField
textField->setMaxLength(10);

textField->addTouchEventListener([&](Ref* sender, Widget::TouchEventType type){
                std::cout << "editing a TextField" << std::endl;
});

this->addChild(textField);
```

7.8 界面

一般使用界面代指 Layer，继承自 Node，用于 Scene 下。

https://blog.csdn.net/lttree/article/details/37997925，Layer 的介绍，不太正式。

https://www.cnblogs.com/as3lib/p/3895843.html，Scene，Layer，Sprite 的关系。

https://www.zhihu.com/question/28233034，为什么在 cocos2d-x 中，大家都喜欢用 Layer 而不是 Scene?

7.9 进阶话题

关于进阶话题，由于介绍已经足够凝练，请自行参考：https://docs.cocos.com/cocos2d-x/manual/zh/#

7.10 调度器

https://blog.csdn.net/u010229677/article/details/14107903，项目中的 scheduleOnce 函数对应文章中对于 schedule 的讲解。

7.11 宏

cocos2dx 中为了简化代码，定义了大量实用的宏。

7.11.1 宏 USING_NS_CC

宏 USING_NS_CC（Using NameSpace CoCos）等价于 using namespace cocos2d。
https://discuss.cocos2d-x.org/t/using-ns-cc-in-header-file/56178,不应在头文件使用宏 USING_NS_CC。