



从零开始的 Cocos2d-x 开发指南

Starting from Scratch: A Cocos2d-x Development Guide

计算机科学与技术学院

林继申 (2250758)

2024 年 12 月 1 日



同濟大學
TONGJI UNIVERSITY



LaTeX Beamer Template

Acknowledgements

本演示文稿由同济大学演示文稿模板（Tongji University Beamer Template）编译。

GitHub 仓库

特别感谢 Federico Zenith 提供的 SINTEF Presentation 模板，以及 Liu Qilong 开发的 Beamer-LaTeX-Themes 派生版本。同时，也感谢 wzl-plasmid 对模板样式的贡献。本项目在这些优秀作品的基础上进行了改进，感谢开源社区所有开发者的支持与贡献。



Table of Contents

1 Cocos2d-x 简介

- ▶ Cocos2d-x 简介
- ▶ Cocos2d-x 基本概念
- ▶ Cocos2d-x 基本功能
- ▶ Cocos2d-x UI 组件
- ▶ Cocos2d-x 进阶内容
- ▶ Cocos2d-x 项目：Teamfight Tactics
- ▶ 网络游戏状态同步



Cocos2d-x 简介

Introduction to Cocos2d-x

- Cocos 产品
 - Cocos Creator
 - Cocos2d-x
 - Cocos Runtime
- 为什么推荐 Cocos2d-x
- 如何上手 Cocos2d-x 引擎



同濟大學
TONGJI UNIVERSITY



Cocos 产品

1 Cocos2d-x 简介

Cocos 是一个开源的游戏开发引擎，提供强大的工具和框架，用于创建跨平台的 2D 和 3D 游戏，支持 C++、JavaScript 和 Lua 等多种编程语言。

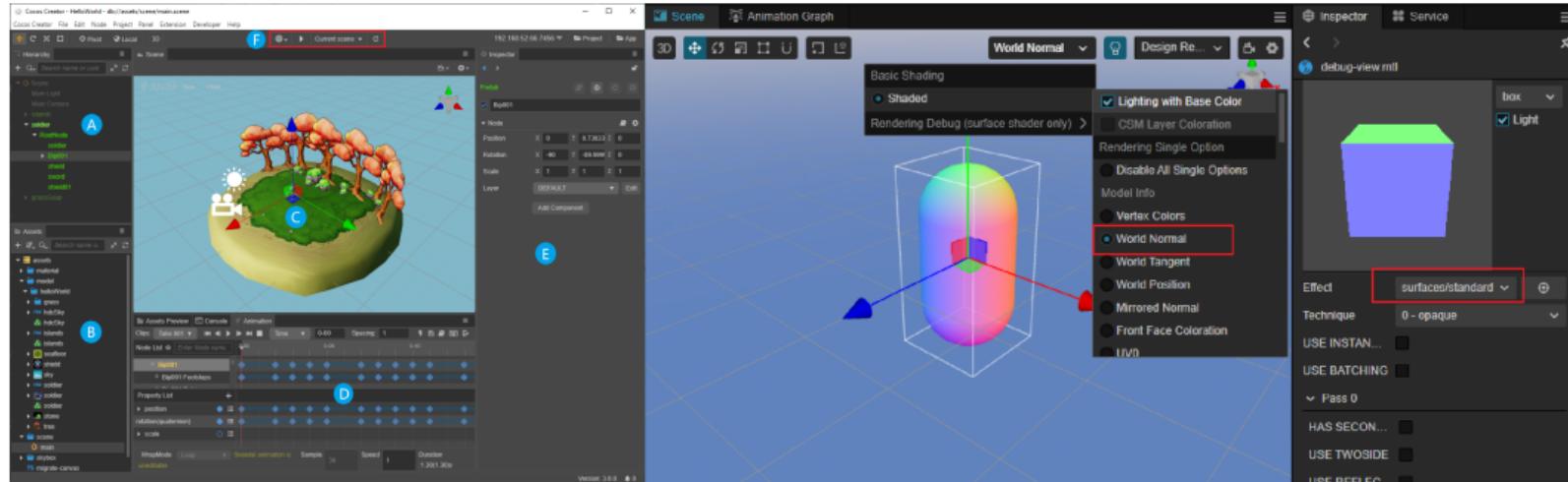
- **Cocos Creator**: 一款高效、轻量开源的跨平台图形引擎，也是一个实时 3D 内容创作平台。
- **Cocos2d-x**: 一个高性能的 C++、Lua、JavaScript 游戏引擎，跨平台支持 iOS、Android 等智能手机，Windows、Mac 等桌面操作系统，以及 Chrome、Safari、IE 等 HTML5 浏览器。
- **Cocos Runtime**: 一个支持互动内容免安装即可运行的商业化 SDK，支持主流 JavaScript 游戏引擎开发的内容，为用户提供数字互动内容“点开即玩”的畅爽体验。



Cocos Creator

1 Cocos2d-x 简介

Cocos Creator：开源、高效、性能卓越的跨平台 3D 实时内容创作引擎。



(Cocos Creator 3.8 用户手册、虚幻引擎 5.5 文档)



Cocos2d-x

1 Cocos2d-x 简介

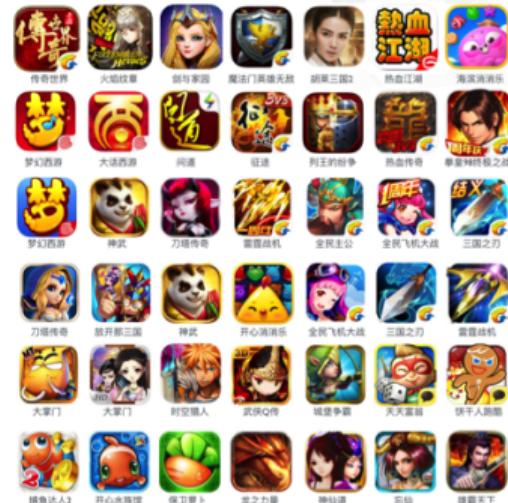
Cocos2d-x：开源、开放、灵活、轻量，即刻帮你拓展边界。

Cocos2d-x 在 2019 年停止更新。

Cocos2d-x 市场占有情况：使用 Cocos2d-x 开发的许多游戏占据苹果应用商店和谷歌应用商店排行榜，同时许多公司如触控、谷歌、微软、ARM，英特尔及的黑莓工程师在 Cocos2d-x 领域也非常活跃。

POWERED BY COCOS

- 2017
- 2016
- 2015
- 2014
- 2013
- 2012





Cocos2d-x

1 Cocos2d-x 简介

Cocos 官网提供了如下学习资源和参考资料：

- [Cocos2d-x 官方文档](#)
- [Cocos2d-x API 文档](#)
- [Cocos2d-x 源代码](#)
- [Cocos2d-x 引擎官方测试项目](#)
- [Cocos2d-x 示例程序](#)

Cocos2d-x 官方文档是最好的入门学习资源，没有之一。

ChatGPT 是最好的 Debug 工具，没有之一。(:D)





Table of Contents

2 Cocos2d-x 基本概念

- ▶ Cocos2d-x 简介
- ▶ Cocos2d-x 基本概念
- ▶ Cocos2d-x 基本功能
- ▶ Cocos2d-x UI 组件
- ▶ Cocos2d-x 进阶内容
- ▶ Cocos2d-x 项目：Teamfight Tactics
- ▶ 网络游戏状态同步



Cocos2d-x 基本概念

Basic Concepts of Cocos2d-x

- 游戏引擎与组件
 - 导演 - 单例模式
 - 场景 - 场景图
 - 精灵
 - 动作与序列
- 节点关系
- 日志输出



同濟大學
TONGJI UNIVERSITY



游戏引擎

2 Cocos2d-x 基本概念

游戏引擎是一种特殊的软件，它提供游戏开发时需要的常见功能；引擎会提供许多组件，使用这些组件能缩短开发时间，让游戏开发变得更容易。

Cocos2d-x 提供了许多易于使用的组件，有着更好的性能，还同时支持移动端和桌面端。Cocos2d-x 通过封装底层图形接口提供了易用的 API，降低了游戏开发的门槛，让使用者可以专注于开发游戏，而不用关注底层的技术细节。（[FastPrinter.cpp](#)、[lib_hdc_tools.cpp](#)）

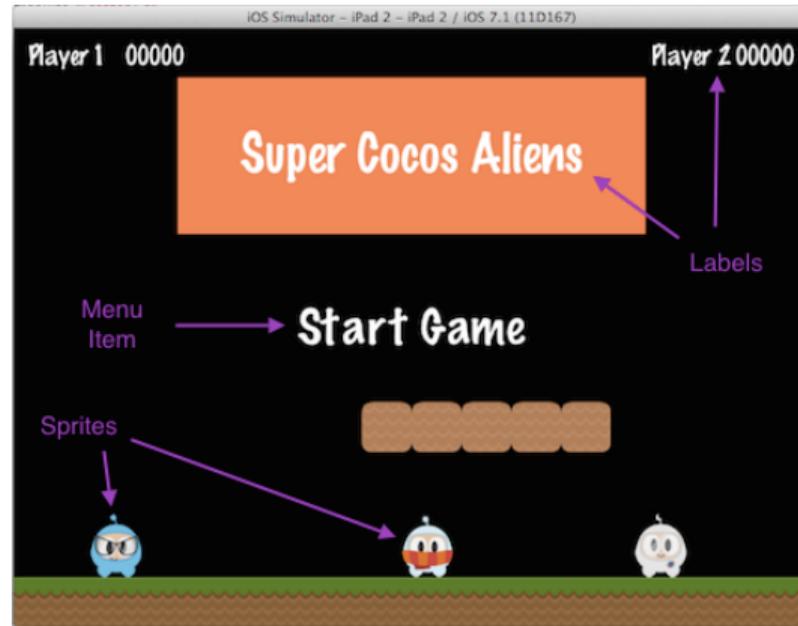
更重要的是 Cocos2d-x 是一个完全开源的游戏引擎，这就允许您在游戏开发过程中根据实际需要，定制化引擎的功能。（[HoverButton](#) 类）



组件

2 Cocos2d-x 基本概念

游戏界面由组件构成。



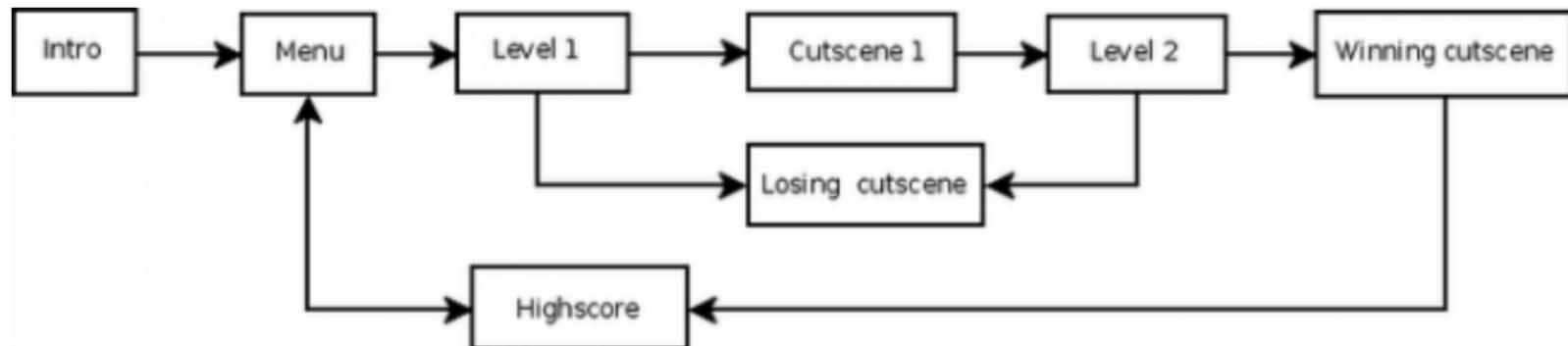


导演 (Director)

2 Cocos2d-x 基本概念

Cocos2d-x 使用导演的概念。在使用 Cocos2d-x 开发游戏的过程中，你可以认为自己是执行制片人，告诉导演该怎么办！一个常见的 Director 任务是控制场景替换和转换。Director 是一个共享的单例对象，可以在代码中的任何地方调用。

这是一个典型的游戏流程实例。当您的游戏设计好时，Director 就负责场景的转换：





单例模式 (Singleton Pattern)

2 Cocos2d-x 基本概念

单例模式 (Singleton Pattern) 是一种设计模式，它的主要目的是确保一个类只有一个实例，并提供一个全局访问点来获取该实例。这样可以保证在整个程序运行期间，某个类的实例不会被多次创建，从而节省系统资源并确保一致性。（[LocationMap 类](#)）

单例模式的核心要点：

- **类的构造函数是私有的**：避免外部代码通过构造函数直接创建多个实例。
- **静态成员变量**：存储唯一的实例。
- **静态成员函数**：提供对外的访问接口，用于获取唯一的实例。



场景 (Scene)

2 Cocos2d-x 基本概念

在游戏开发过程中，你可能需要一个主菜单，几个关卡和一个结束场景。如何组织所有这些分开的部分？使用场景（Scene）！（[Scene 类](#)）

场景是由很多小的对象拼接而成，所有的对象组合在一起，形成了最终的结果。场景是被渲染器（renderer）画出来的。渲染器负责渲染精灵和其他的对象进入屏幕。为了更好的理解这个过程，我们需要讨论一下场景图。



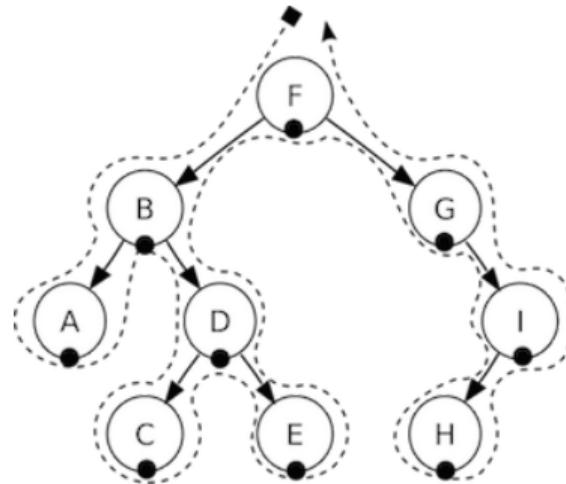
场景图 (Scene Graph)

2 Cocos2d-x 基本概念

场景图 (Scene Graph) 是一种安排场景内对象的数据结构，它把场景内所有的节点 (Node) 都包含在一个树 (Tree) 上。（场景图虽然叫做“图”，但实际使用一个树结构来表示）。

当你开发游戏的时候，你会添加一些节点，精灵和动画到一个场景中，你期望的是每一个添加的对象都能被正确的展示，可是如果有对象没有被展示呢？可能你错误的把这个对象隐藏到背景中了。

既然场景图是一个树结构，你就能遍历它，Cocos2d-x 使用中序遍历，先遍历左子树，然后根节点，最后是右子树。中序遍历下图的节点，能得到 A, B, C, D, E, F, G, H, I 这样的序列。





场景图 (Scene Graph)

2 Cocos2d-x 基本概念

在 Cocos2d-x 中，通过 Scene 的 `addChild()` 方法构建场景图。

渲染时 z-order 值大的节点对象会后绘制，值小的节点对象先绘制。如果两个节点对象的绘制范围有重叠，z-order 值大的可能会覆盖 z-order 值小的。

添加场景

```
scene->addChild(title_node, -2);  
  
scene->addChild(label_node);  
  
scene->addChild(sprite_node, 1);
```



精灵 (Sprite)

2 Cocos2d-x 基本概念

精灵是能在屏幕上移动的对象，它能被控制。

每个图形对象都是一个精灵吗？不是的。为什么？如果你能控制它，它才是一个精灵，如果无法控制，那就只是一个节点（Node）。

Sprite 很容易被创建，它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。



精灵 (Sprite)

2 Cocos2d-x 基本概念

配置位置、旋转角度、缩放比例

```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setPosition(Vec2(500, 0));
mySprite->setRotation(40);
mySprite->setScale(2.0);
```





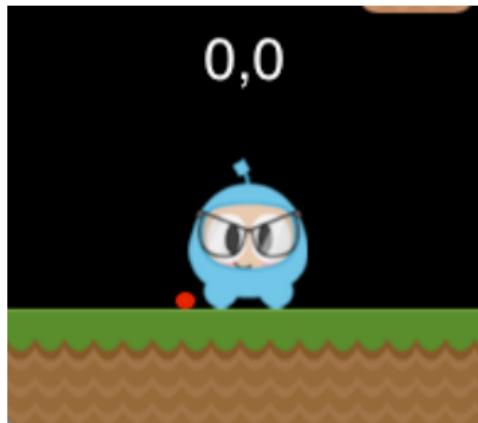
精灵 (Sprite)

2 Cocos2d-x 基本概念

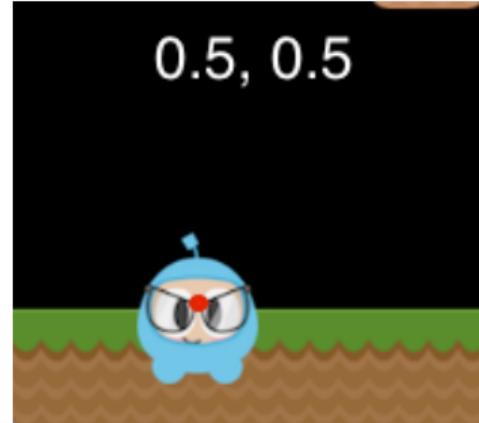
配置锚点

```
mySprite->setAnchorPoint(Vec2(0, 0));  
mySprite->setAnchorPoint(Vec2(0.5, 0.5));  
mySprite->setAnchorPoint(Vec2(1, 1));
```

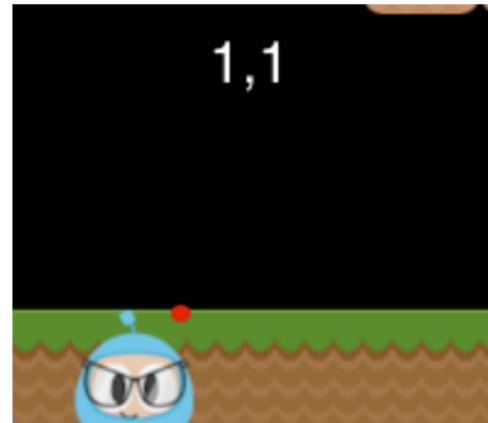
0,0



0.5, 0.5



1,1





动作 (Action)

2 Cocos2d-x 基本概念

动作 (Action) 可以让精灵在场景中移动，如从一个点移动到另外一个点。

执行动作

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

```
// 创建 MoveBy，让精灵在 2 秒内向右移动 50 像素，向上移动 10 像素
auto moveBy = MoveBy::create(2, Vec2(50,10));
mySprite->runAction(moveBy);
```

```
// 创建 MoveTo，让精灵在 2 秒内移动到指定位置 (50, 10)
auto moveTo = MoveTo::create(2, Vec2(50,10));
mySprite->runAction(moveTo);
```



序列 (Sequence)

2 Cocos2d-x 基本概念

序列 (Sequence) 支持执行多个 Action。

执行动作序列

```
auto mySprite = 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));
auto delay = DelayTime::create(1);

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



序列 (Sequence)

2 Cocos2d-x 基本概念

通过引擎中的 Spawn 对象，能让多个动作同时被解析执行。可能不同动作的执行时间不一致，在这种情况下，他们不会同时结束。

同时执行多个动作

```
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));
```



节点关系

2 Cocos2d-x 基本概念

Cocos2d-x 的节点关系，是被附属和附属的关系，就像数据结构中的父子关系，如果两个节点被添加到一个父子关系中，那么父节点的属性变化会被自动应用到子节点中。

需要注意的是，父节点的属性（如位置、缩放、旋转等）会影响到子节点的变换效果（例如子节点的位置、大小和旋转角度等），但是父节点的锚点不会直接改变子节点的锚点。子节点的锚点总是相对于子节点自身的坐标系独立存在，不会被父节点的锚点所影响。父节点的锚点变动会影响子节点在父节点坐标系中的位置变换，但不会影响子节点本身的锚点。



日志输出

2 Cocos2d-x 基本概念

有时，在你的游戏正在运行的时候，为了了解程序的运行过程或是为了查找一个 Bug，你想看到一些运行时信息，可以使用 `log()` 把信息输出到控制台。

日志输出

```
log("This would be outputted to the console");
log("string is %s", s);
log("double is %f", d);
log("integer is %d", i);
log("float is %f", f);
```

对于使用 C++ 进行游戏开发的用户来说，可能想使用 `std::cout` 而不用 `log()`，实际上 `log()` 更易于使用，它格式化复杂的输出信息更简单。



Table of Contents

3 Cocos2d-x 基本功能

- ▶ Cocos2d-x 简介
- ▶ Cocos2d-x 基本概念
- ▶ Cocos2d-x 基本功能
- ▶ Cocos2d-x UI 组件
- ▶ Cocos2d-x 进阶内容
- ▶ Cocos2d-x 项目：Teamfight Tactics
- ▶ 网络游戏状态同步



Cocos2d-x 基本功能

Basic Features of Cocos2d-x

- 基本功能
 - 精灵 - Sprite 类方法
 - 动作 - 动作序列分析
 - 场景 - 场景创建与场景切换
- 如何查阅官方文档



同濟大學
TONGJI UNIVERSITY



精灵 (Sprite)

3 Cocos2d-x 基本功能

在[Cocos2d-x 官方文档](#)中，关于精灵部分给出了以下内容：

- **精灵创建**: 使用图像创建、使用图集创建、使用精灵缓存创建
- **精灵控制**: 锚点、位置、旋转、缩放、倾斜、颜色、透明度
- **多边形精灵**: 为什么要使用多边形精灵、AutoPolygon

阅读官方文档是程序开发者的必备能力。

完整精灵 (Sprite) 类方法在[Cocos2d-x API 文档](#)中获取。



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

公共成员函数：

- `virtual bool isFrameDisplayed (SpriteFrame *frame) const`
返回一个布尔值，表示当前是否正在显示指定的 SpriteFrame
- `virtual SpriteFrame *getSpriteFrame () const`
返回当前显示的帧
- `V3F_C4B_T2F_Quad getQuad () const`
返回精灵的四边形（纹理坐标、顶点坐标和颜色）信息
- `bool isTextureRectRotated () const`
返回一个布尔值，表示纹理矩形是否被旋转
- `unsigned int getAtlasIndex () const`
返回在纹理图集中的索引
- `void setAtlasIndex (unsigned int atlasIndex)`
设置纹理图集的索引



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

公共成员函数：

- const Rect &getTextureRect () const
返回精灵的纹理矩形
- TextureAtlas *getTextureAtlas () const
获取精灵渲染时所使用的纹理图集的弱引用
- virtual void setProgramState (backend::ProgramState *programState) override
设置程序状态
- virtual backend::ProgramState *getProgramState () const override
获取当前的程序状态
- void setTextureAtlas (TextureAtlas *textureAtlas)
设置精灵渲染时使用的纹理图集的弱引用
- const Vec2 &getOffsetPosition () const
获取精灵的偏移位置



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

公共成员函数：

- `bool isFlippedX () const`
返回一个布尔值，表示精灵是否水平翻转
- `void setFlippedX (bool flippedX)`
设置精灵是否水平翻转
- `bool isFlippedY () const`
返回一个布尔值，表示精灵是否垂直翻转
- `void setFlippedY (bool flippedY)`
设置精灵是否垂直翻转
- `const PolygonInfo &getPolygonInfo () const`
返回与精灵关联的多边形信息的引用
- `void setPolygonInfo (const PolygonInfo &info)`
设置精灵使用的新多边形信息



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

公共成员函数：

- void setStretchEnabled (bool enabled)
设置是否启用内容大小拉伸纹理
- bool isStretchEnabled () const
返回一个布尔值，表示内容大小是否拉伸纹理

批处理节点方法：

- virtual void updateTransform () override
根据旋转、位置和缩放值更新四边形
- virtual SpriteBatchNode *getBatchNode () const
返回精灵渲染时使用的批处理节点对象
- virtual void setBatchNode (SpriteBatchNode *spriteBatchNode)
设置批处理节点



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

纹理 / 帧方法：

- `virtual void setTexture (const std::string &filename)`
设置一个新的纹理（通过文件名）
- `virtual void setTexture (Texture2D *texture) override`
设置新的纹理（通过 Texture2D 对象），并且纹理的矩形不会改变
- `virtual Texture2D *getTexture () const override`
返回精灵使用的 Texture2D 对象
- `virtual void setTextureRect (const Rect &rect)`
更新精灵的纹理矩形
- `virtual void setTextureRect (const Rect &rect, bool rotated,
const Size &untrimmedSize)`
通过不同的参数更新纹理矩形，包括是否旋转和未裁剪的尺寸
- `virtual void setVertexRect (const Rect &rect)`
设置顶点矩形



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

纹理 / 帧方法：

- `virtual void setCenterRectNormalized (const Rect &rect)`
设置规范化的中心矩形
- `virtual Rect getCenterRectNormalized () const`
获取规范化的中心矩形
- `virtual void setCenterRect (const Rect &rect)`
设置中心矩形
- `virtual Rect getCenterRect () const`
返回 Cap Insets 矩形
- `virtual void setSpriteFrame (const std::string &spriteFrameName)`
设置精灵帧（通过名称）
- `virtual void setSpriteFrame (SpriteFrame *newFrame)`
设置精灵帧（通过 SpriteFrame 对象）



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

动画方法：

- `virtual void setDisplayFrameWithAnimationName (const std::string &animationName, unsigned int frameIndex)`
根据动画名称和帧索引设置当前显示的帧

精灵属性的设置器 / 获取器：

- `virtual bool isDirty () const`
返回一个布尔值，表示精灵是否需要在图集中更新
- `virtual void setDirty (bool dirty)`
设置精灵为脏状态，表示需要在图集中更新
- `virtual std::string getDescription () const override`
获取精灵的描述信息



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

从 Node 类继承的函数：

- `virtual void setScaleX (float scaleX) override`
设置节点的水平缩放
- `virtual void setScaleY (float scaleY) override`
设置节点的垂直缩放
- `virtual void setScale (float scaleX, float scaleY) override`
设置节点的水平和垂直缩放
- `virtual void setPosition (const Vec2 &pos) override`
设置节点在父节点坐标系统中的位置
- `virtual void setPosition (float x, float y) override`
设置节点在父节点坐标系统中的位置
- `virtual void setRotation (float rotation) override`
设置节点的旋转角度（以度为单位）



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

从 Node 类继承的函数：

- `virtual void setRotationSkewX (float rotationX) override`
设置节点的水平旋转（以度为单位）
- `virtual void setRotationSkewY (float rotationY) override`
设置节点的垂直旋转（以度为单位）
- `virtual void setSkewX (float sx) override`
设置节点的水平倾斜角度（以度为单位）
- `virtual void setSkewY (float sy) override`
设置节点的垂直倾斜角度（以度为单位）
- `virtual void removeChild (Node *child, bool cleanup) override`
从容器中移除子节点
- `virtual void removeAllChildrenWithCleanup (bool cleanup) override`
移除容器中的所有子节点，并根据 cleanup 参数进行清理



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

从 Node 类继承的函数：

- virtual void reorderChild (Node *child, int zOrder) override
根据新的 z 值重新排序子节点
- virtual void addChild (Node *child, int zOrder, int tag) override
将子节点添加到容器中，并指定 z 顺序和标签
- virtual void addChild (Node *child, int zOrder, const std::string &name) override
将子节点添加到容器中，并指定 z 顺序和名称
- virtual void sortAllChildren () override
在绘制之前对子节点进行排序
- virtual void setScale (float scale) override
设置节点的缩放
- virtual void setPositionZ (float positionZ) override
设置节点的 Z 坐标



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

从 Node 类继承的函数：

- virtual void setAnchorPoint (const Vec2 &anchor) override
设置节点的锚点（以百分比表示）
- virtual void setContentSize (const Size &size) override
设置节点的原始大小
- virtual void setIgnoreAnchorPointForPosition (bool value) override
设置是否忽略锚点来定位节点
- virtual void setVisible (bool bVisible) override
设置节点的可见性
- virtual void draw (Renderer *renderer, const Mat4 &transform, uint32_t flags) override
重写此方法来绘制自定义节点



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

从 Node 类继承的函数：

- `virtual void setOpacityModifyRGB (bool modify) override`
如果需要透明度影响颜色属性，则设置为 true
- `virtual bool isOpacityModifyRGB () const override`
如果透明度会影响 RGB 颜色值，则返回 true
- `virtual void addChild (Node *child)`
将子节点添加到容器中，默认 z 顺序为 0
- `virtual void addChild (Node *child, int localZOrder)`
将子节点添加到容器中，并指定局部 z 顺序
- `virtual void addChild (Node *child, int localZOrder, int tag)`
将子节点添加到容器中，并指定局部 z 顺序和标签



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

从 Node 类继承的函数：

- `virtual void addChild (Node *child, int localZOrder, const std::string &name)`

将子节点添加到容器中，并指定局部 z 顺序和名称

从 TextureProtocol 类继承的函数：

- `void setBlendFunc (const BlendFunc &blendFunc) override`
设置混合函数
- `const BlendFunc &getBlendFunc () const override`
获取混合函数



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

Creators:

- static Sprite *create ()
创建一个空的精灵，不带纹理
- static Sprite *create (const std::string &filename)
通过图像文件名创建精灵
- static Sprite *create (const PolygonInfo &info)
通过多边形信息创建一个多边形精灵
- static Sprite *create (const std::string &filename, const Rect &rect)
通过图像文件名和矩形区域创建精灵



精灵 (Sprite) 类方法

3 Cocos2d-x 基本功能

Creators:

- static Sprite *createWithTexture (Texture2D *texture)
通过 Texture2D 对象创建精灵
- static Sprite *createWithTexture (Texture2D *texture, const Rect &rect, bool rotated=false)
通过 Texture2D 对象和矩形区域创建精灵（可选旋转）
- static Sprite *createWithSpriteFrame (SpriteFrame *spriteFrame)
通过精灵帧创建精灵
- static Sprite *createWithSpriteFrameName (const std::string &spriteFrameName)
通过精灵帧名称创建精灵



查阅官方文档

3 Cocos2d-x 基本功能

- 官方文档是掌握工具或框架的核心特性和最佳实践
- 查阅官方文档能够获取最新版本功能、改进和修复
- 官方文档提供详尽的 API、FAQ 和代码示例
- 官方文档避免低效或错误实现
- 查阅官方文档提高独立问题解决能力
- 查阅官方文档减少对他人依赖
- 官方文档统一技术标准，提升团队工作效率



动作 (Actions)

3 Cocos2d-x 基本功能

基本动作：

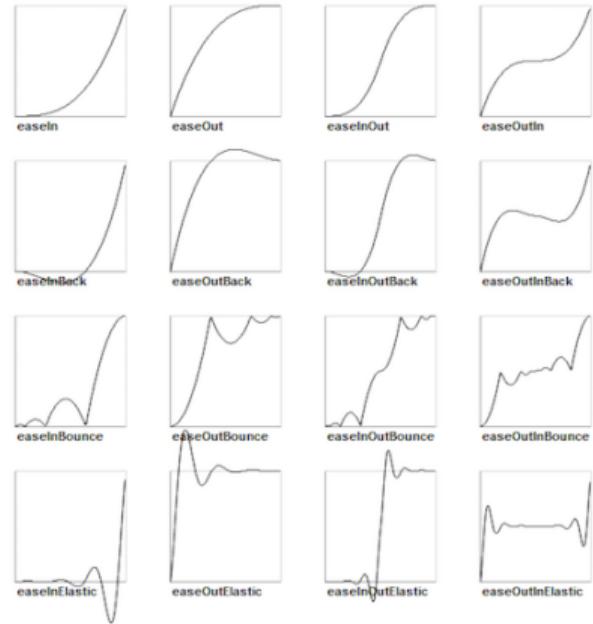
- 使用 MoveTo 和 MoveBy 完成节点对象在一个设置的时间后移动
- 使用 RotateTo 和 RotateBy 完成节点对象在一个设置的时间后顺时针旋转指定角度
- 使用 ScaleBy 和 ScaleTo 完成节点对象的比例缩放
- 使用 FadeIn 和 FadeOut 完成节点对象的淡入淡出
- 使用 TintTo 和 TintBy 将一个实现了 NodeRGB 协议的节点对象进行色彩混合



动作 (Actions)

3 Cocos2d-x 基本功能

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。当然你也可以将变速动作应用到动画菜单和按钮上，实现你想要的效果。

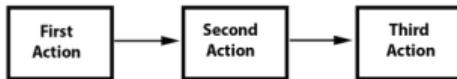




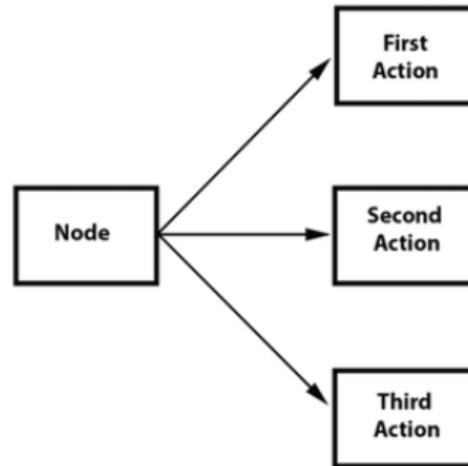
序列 (Sequence)

3 Cocos2d-x 基本功能

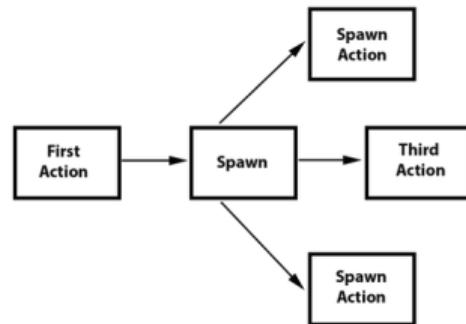
动作序列 (Sequence) 是一种封装多个动作的对象，当这个对象执行时被封装的动作会顺序执行。



Spawn 和 Sequence 是非常相似的，区别是 Spawn 同时执行所有的动作。



可以把一个 Spawn 添加到一个 Sequence 中。





动作的克隆

3 Cocos2d-x 基本功能

克隆 (Clone) 的功能和字面含义一样，如果你对一个节点对象使用了 `clone()` 方法，你就获得了这个节点对象的拷贝。

为什么要使用 `clone()` 方法？因为当 Action 对象运行时会产生一个内部状态，记录着节点属性的改变。当你想将一个创建的动作，重复使用到不同的节点对象时，如果不用 `clone()` 方法，就无法确定这个动作的属性到底是怎样的（因为被使用过，产生了内部状态），这会造成难以预料的结果。



动作的克隆

3 Cocos2d-x 基本功能

我们来看示例，假如你有一个坐标位置是 (0,0) 的 heroSprite，执行这样一个动作：

示例动作

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

你的 heroSprite 就在 10s 的时间中，从 (0,0) 移动到了 (400,100)，heroSprite 有了一个新位置 (400,100)，更重要的是动作对象也有了节点位置相关的内部状态了。现在假如你有一个坐标位置是 (200,200) 的 enemySprite。你还使用这个相同动作，enemySprite 就会移动到 (800,200) 的坐标位置，并不是你期待的结果。因为第二次将这个动作应用的时候，它已经有内部状态了。使用 clone() 能避免这种情况，克隆获得一个新的动作对象，新的对象没有之前的内部状态。



动作的克隆

3 Cocos2d-x 基本功能

错误示例

```
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); // oops, this will not be unique!
// uses the Actions current internal state as a starting point.
```



动作的克隆

3 Cocos2d-x 基本功能

正确示例

```
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.
```



动作的倒转

3 Cocos2d-x 基本功能

倒转 (Reverse) 的功能也和字面意思一样，调用 `reverse()` 可以让一系列动作按相反的方向执行。`reverse()` 不是只能简单的让一个 Action 对象反向执行，还能让 Sequence 和 Spawn 倒转。

示例动作

```
mySprite->runAction(mySpawn->reverse());
```



动作序列分析

3 Cocos2d-x 基本功能

动作序列分析

```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setPosition(50, 56);
auto moveBy = MoveBy::create(2.0f, Vec2(500,0));
auto scaleBy = ScaleBy::create(2.0f, 2.0f);
auto delay = DelayTime::create(2.0f);

auto delaySequence = Sequence::create(delay, delay->clone(),
                                      delay->clone(), nullptr);
auto sequence = Sequence::create(moveBy, delay, scaleBy,
                                 delaySequence, nullptr);
mySprite->runAction(sequence);
mySprite->runAction(sequence->reverse());
```



场景创建

3 Cocos2d-x 基本功能

场景创建

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

auto myScene = Scene::create();

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

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



场景切换方式

3 Cocos2d-x 基本功能

场景切换方式

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

Director::getInstance()->runWithScene(myScene);
Director::getInstance()->replaceScene(myScene);
Director::getInstance()->pushScene(myScene);
Director::getInstance()->popScene();
```



场景切换方式

3 Cocos2d-x 基本功能

- `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。



场景切换效果

3 Cocos2d-x 基本功能

场景切换效果

// 漫变过渡

```
Director::getInstance()->replaceScene(  
    TransitionFade::create(0.5, myScene, Color3B(0,255,255)));
```

// X 轴翻转过渡

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

// 从顶部滑入过渡

```
Director::getInstance()->replaceScene(  
    TransitionSlideInT::create(1, myScene));
```



Table of Contents

4 Cocos2d-x UI 组件

- ▶ Cocos2d-x 简介
- ▶ Cocos2d-x 基本概念
- ▶ Cocos2d-x 基本功能
- ▶ Cocos2d-x UI 组件
- ▶ Cocos2d-x 进阶内容
- ▶ Cocos2d-x 项目：Teamfight Tactics
- ▶ 网络游戏状态同步



Cocos2d-x UI 组件

Cocos2d-x UI Components

Cocos2d-x 提供了一套易用的
UI 组件，游戏开发过程中，你
能很容易的把它们添加到游
戏中。



同濟大學
TONGJI UNIVERSITY



标签 (Label)

4 Cocos2d-x UI 组件

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。

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



创建 BMFont 标签

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



标签 (Label)

4 Cocos2d-x UI 组件

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。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。

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



创建 TTF 标签

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



标签 (Label)

4 Cocos2d-x UI 组件

如果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。

创建 TTFConfig 对象

```
TTFConfig labelConfig; // 创建配置
labelConfig.fontFilePath = "myFont.ttf"; // 字体文件
labelConfig.fontSize = 16; // 字体大小
labelConfig.glyphs = GlyphCollection::DYNAMIC; // 使用动态字形
labelConfig.outlineSize = 0; // 无外轮廓
labelConfig.customGlyphs = nullptr; // 无自定义字形
labelConfig.distanceFieldEnabled = false; // 禁用距离场渲染
// 使用 TTFConfig 配置创建标签
auto myLabel = Label::createWithTTF(labelConfig, "Text");
```



标签 (Label)

4 Cocos2d-x UI 组件

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。

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



创建 SystemFont 标签

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



标签 (Label)

4 Cocos2d-x UI 组件

标签可以添加阴影效果。

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



创建阴影效果标签

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



标签 (Label)

4 Cocos2d-x UI 组件

标签可以添加描边效果。

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



创建描边效果标签

```
auto myLabel = Label::createWithTTF("myFont.ttf", "Text", 16);
myLabel->enableOutline(Color4B::WHITE, 1);
```



标签 (Label)

4 Cocos2d-x UI 组件

标签可以添加发光效果。

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



创建发光效果标签

```
auto myLabel = Label::createWithTTF("myFont.ttf", "Text", 16);  
myLabel->enableGlow(Color4B::YELLOW);
```



菜单 (Menu)

4 Cocos2d-x UI 组件

创建菜单

```
// 创建一个存储菜单项的 Vector  
Vector<MenuItem*> MenuItems;
```

```
// 创建一个关闭菜单项，使用图片并绑定回调函数  
auto closeItem = MenuItemImage::create("Normal.png", "Selected.png",  
    CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));
```

```
// 将关闭菜单项添加到 MenuItems 向量中  
MenuItems.pushBack(closeItem);
```

```
/* 如果有更多的菜单项，可以继续类似地创建并添加到 MenuItems 向量中 */
```



菜单 (Menu)

4 Cocos2d-x UI 组件

创建菜单

```
/* 如果有更多的菜单项，可以继续类似地创建并添加到 MenuItems 向量中 */
auto anotherItem = MenuItemImage::create("Normal.png", "Selected.png",
    CC_CALLBACK_1(HelloWorld::anotherCallback, this));
MenuItems.pushBack(anotherItem);

// 创建菜单，并将 MenuItems 向量传递给它
auto menu = Menu::createWithArray(MenuItems);

// 将菜单添加到当前节点（假设当前节点是 this），设置层级为 1
this->addChild(menu, 1);
```



按钮 (Button)

4 Cocos2d-x UI 组件

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。[\(HoverButton 类\)](#)

在屏幕显示的时候，同一个时刻只能看到一个状态，正常显示状态像这样：





按钮 (Button)

4 Cocos2d-x UI 组件

创建按钮

```
auto button = Button::create("normal_image.png",
                             "selected_image.png",
                             "disabled_image.png");
button->setTitleText("Button Text");
button->addTouchEventListener([&](Ref* sender,
                                    Widget::TouchEvent type) {
    switch (type) {
        case ui::Widget::TouchEvent::BEGAN: ...
        case ui::Widget::TouchEvent::ENDED: ...
        default: break;
    }
});
```



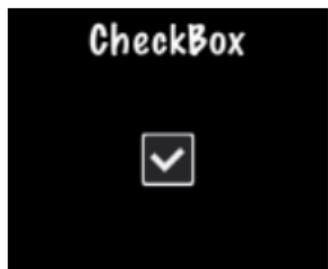
复选框 (CheckBox)

4 Cocos2d-x UI 组件

只有两种状态的项目经常被设计为复选框。我们为一个复选框指定了五张图像，因为复选框有五种状态：未被选中，被点击，未被选中时不可用，被选中，选中时不可用。这样五种状态的图像依次如下：



在屏幕显示的时候，同一个时刻只能看到一个状态，被选中时状态像这样：





复选框 (CheckBox)

4 Cocos2d-x UI 组件

创建复选框

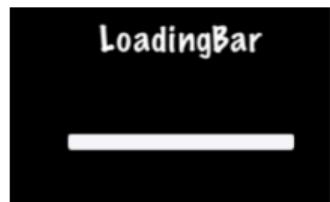
```
auto checkbox = CheckBox::create("check_box_normal.png",
    "check_box_normal_press.png", "check_box_active.png",
    "check_box_normal_disable.png", "check_box_active_disable.png");
checkbox->addTouchEventListener([&] (Ref* sender,
                                         Widget::TouchEvent type) {
    switch (type) {
        case ui::Widget::TouchEvent::BEGAN: ...
        case ui::Widget::TouchEvent::ENDED: ...
        default: ...
    }
});
```



进度条 (LoadingBar)

4 Cocos2d-x UI 组件

进度条当进度增加时，进度条向右填充。在屏幕上一个满进度的进度条是这样的：



创建进度条

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

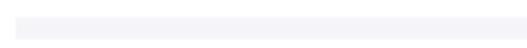
loadingBar->setPercent(25);
loadingBar->setPercent(35);
```



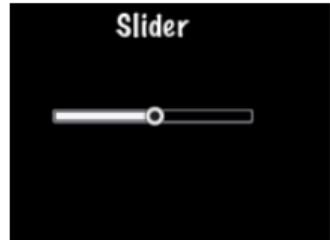
滑动条 (Slider)

4 Cocos2d-x UI 组件

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。



在屏幕上一个滑动条看起来是这样的：





滑动条 (Slider)

4 Cocos2d-x UI 组件

创建滑动条

```
auto slider = Slider::create();
slider->loadBarTexture("Slider_Back.png");
slider->loadSlidBallTextures("Normal.png", "Press.png", "Disable.png");
slider->loadProgressBarTexture("Slider_PressBar.png");
slider->addTouchEventListener([&] (Ref* sender,
                                         Widget::TouchEvent type) {
    switch (type) {
        case ui::Widget::TouchEvent::BEGAN: ...
        case ui::Widget::TouchEvent::ENDED: ...
        default: ...
    }
});
```



文本框 (TextField)

4 Cocos2d-x UI 组件

Cocos2d-x 提供 TextField 满足文本输入需求。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。

创建文本框

```
auto textField = TextField::create("", "Arial", 30);
textField->setPasswordEnabled(true);
textField->setMaxLength(10);
textField->addTouchEventListener([&](Ref* sender,
                                         Widget::TouchEvent type) {
    std::cout << "Editing a TextField" << std::endl;
});
```



Table of Contents

5 Cocos2d-x 进阶内容

- ▶ Cocos2d-x 简介
- ▶ Cocos2d-x 基本概念
- ▶ Cocos2d-x 基本功能
- ▶ Cocos2d-x UI 组件
- ▶ Cocos2d-x 进阶内容
- ▶ Cocos2d-x 项目：Teamfight Tactics
- ▶ 网络游戏状态同步



Cocos2d-x 进阶内容

The Advanced Content of Cocos2d-x

- 瓦片地图
- 粒子系统
- 视差滚动
- 事件分发机制



同濟大學
TONGJI UNIVERSITY



瓦片地图

5 Cocos2d-x 进阶内容

在游戏开发过程中，我们会遇到超过屏幕大小的地图，例如在即时战略游戏中，它使得玩家可以在地图中滚动游戏画面。这类游戏通常会有丰富的背景元素，如果直接使用背景图切换的方式，需要为每个不同的场景准备一张背景图，而且每个背景图都不小，这样会造成资源浪费。

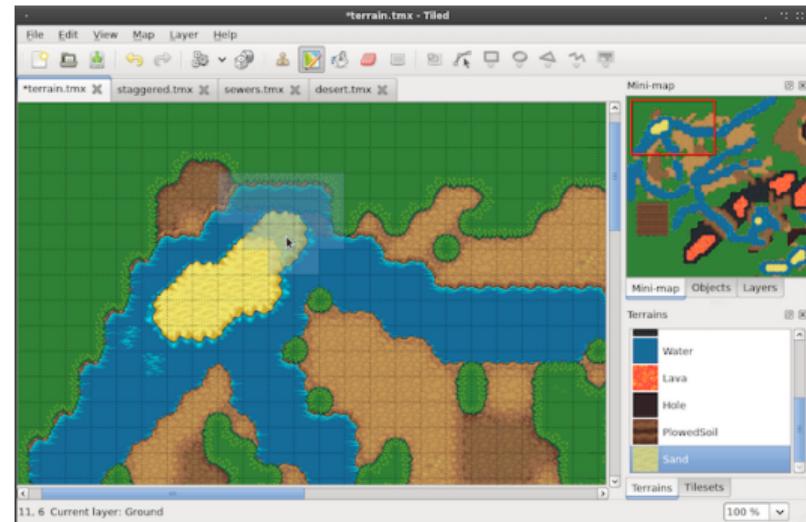
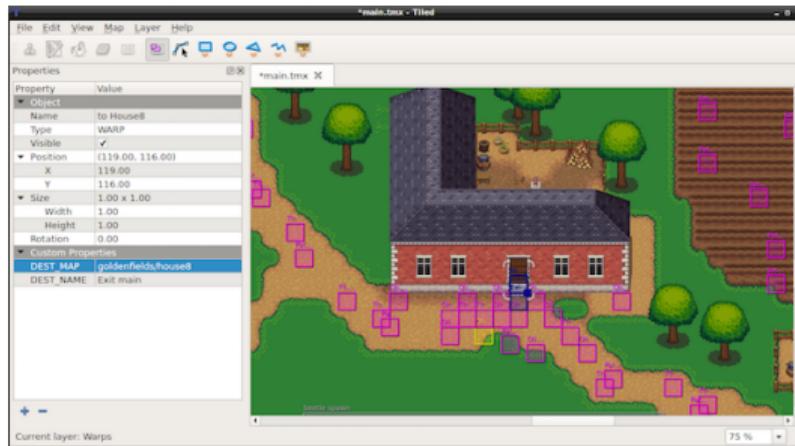
瓦片地图就是为了解决这问题而产生的。一张大的世界地图或者背景图可以由几种地形来表示，每种地形对应一张小的图片，我们称这些小的地形图片为瓦片。把这些瓦片拼接在一起，一个完整的地图就组合出来了，这就是瓦片地图的原理。



瓦片地图

5 Cocos2d-x 进阶内容

在 Cocos2d-x 中，瓦片地图实现的是 TileMap 方案，TileMap 要求每个瓦片占据地图上一个四边形或六边形的区域。把不同的瓦片拼接在一起，就可以组成完整的地图。TileMap 使用一种基于 XML 的 TMX 格式文件。





粒子系统

5 Cocos2d-x 进阶内容

粒子系统是指计算机图形学中模拟特定现象的技术，它在模仿自然现象、物理现象及空间扭曲上具备得天独厚的优势，能为我们实现一些真实自然而又带有随机性的效果（如爆炸、烟花、水流）提供了方便。Cocos2d-x 引擎中就为我们提供了强大的粒子系统。



视差滚动

5 Cocos2d-x 进阶内容

视差滚动是指让多层背景以不同的速度移动，从而形成的立体运动效果。比如超级马里奥游戏中，角色所在地面的移动与背景天空的移动，就是一个视差滚动。

Cocos2d-x 通过 ParallaxNode 对象模拟视差滚动。可以通过序列控制移动，也可以通过监听鼠标，触摸，加速度计，键盘等事件控制移动。ParallaxNode 对象比常规节点对象复杂一些，因为为了呈现不同的移动速度，需要多个子节点。它类似 Menu 像一个容器，本身不移动，移动的是被添加进入其中的不同子节点。



事件分发机制

5 Cocos2d-x 进阶内容

Cocos2d-x 通过事件分发机制响应用户事件，已内置支持常见的事件如触摸事件，键盘事件等。同时提供了创建自定义事件的方法，满足我们在游戏的开发过程中，特殊的事件响应需求。

基本元素：

- **事件监听器**：负责接收事件，并执行预定义的事件处理函数
- **事件分发器**：负责发起通知
- **事件对象**：记录事件的相关信息



Table of Contents

6 Cocos2d-x 项目：Teamfight Tactics

- ▶ Cocos2d-x 简介
- ▶ Cocos2d-x 基本概念
- ▶ Cocos2d-x 基本功能
- ▶ Cocos2d-x UI 组件
- ▶ Cocos2d-x 进阶内容
- ▶ Cocos2d-x 项目：Teamfight Tactics
- ▶ 网络游戏状态同步



Cocos2d-x 项目：Teamfight Tactics

Cocos2d-x Project: Teamfight Tactics

基于 Cocos2d-x 3.17.2 开发的
金铲铲之战游戏项目（2023
年同济大学程序设计范式课
程项目）



同濟大學
TONGJI UNIVERSITY



Teamfight Tactics

6 Cocos2d-x 项目：Teamfight Tactics

基于 Cocos2d-x 3.17.2 开发的金铲铲之战游戏项目。

GitHub 仓库

该项目是一个基于 Cocos2d-x 3.17.2 开发的金铲铲之战游戏，灵感来自于《Dota 自走棋》《云顶之弈》《金铲铲之战》等游戏。游戏拥有丰富的基础功能，如具有引人入胜的初始和设置界面、背景音效、多种卡牌及其升级系统、小小英雄的移动以及带有红蓝血条的场上卡牌，卡牌蓝条满时还能释放技能。此外，游戏支持创建和加入房间，提供练习模式和联机对弈，允许玩家与 AI 或其他人类玩家进行游戏。项目还扩展了多种羁绊的加强功能和强化符文系统，以及战斗中的音效，为玩家带来更加丰富和沉浸式的游戏体验。



Teamfight Tactics

6 Cocos2d-x 项目：Teamfight Tactics





Teamfight Tactics

6 Cocos2d-x 项目：Teamfight Tactics





Teamfight Tactics

6 Cocos2d-x 项目：Teamfight Tactics

8 收件人: 你

周日 2024/1/7 20:31

您好：
在Github上发现了你们的Teamfight Tactics 项目。我非常感兴趣，请问你们有计划做成一个商业化的项目吗？非常希望能有交流群可以交流一下。

Lin Minmus
收件人: 你

周日 2024/1/7 21:23

您好！
非常感谢您对我们 Teamfight Tactics 项目的关注，我们收到您的邮件时非常激动和欣喜。您的兴趣和认可对我们来说意义重大，这是我们项目开发过程中第一次收到如此积极的反馈，对我们而言是极大的鼓励。

尽管如此，由于我们目前还是本科二年级学生，学业繁忙且能力有限，我们暂时没有计划将这个项目商业化。同时，我们也没有建立相关交流群，主要是因为目前无法分配更多时间和精力来管理和维护。

我们对于不能与您更深入交流和合作感到遗憾。但请相信，您的邮件对我们来说极具鼓舞意义，我们会将这份激情和动力转化为进一步学习和成长的动力。

感谢您的理解和支持，期待未来有更多的机会与您交流！

祝好！

林继申 同济大学软件学院



Teamfight Tactics

6 Cocos2d-x 项目：Teamfight Tactics

回复：回复：关于Teamfight_Tactics



...@qq.com

您好：在Github上发现了你们的Teamfight Tactics 项目，我非常感兴趣，请问你们有计划做一个商业化项目吗？非常...

周日 2024/1/7 20:31



Lin Minmus

您好！非常感谢您对我们 Teamfight Tactics 项目的关注，我们收到您的邮件时非常激动和欣喜。您的兴趣和认可对我们来...

周日 2024/1/7 21:23



...@qq.com <...@qq.com>

...

周日 2024/1/7 21:48

林同学：

没想到能这么快收到你的回信，确实学业对你们现在来说非常重要。加油。

我也是一个有20多年研发经验的老程序员了，我主攻方向是构架和微服务（golang语言方面），有空可以多交流，需要我支持可以直接微信我。

我的微信ID：...@wxid...

祝你们学业有成，项目也越来越好。



...

谢谢，我会的。

我会与你保持联络。

再次表示感谢！

↪ 答复

↪ 转发



Teamfight Tactics

6 Cocos2d-x 项目：Teamfight Tactics

请问github金铲铲之战项目如何使用

收件人: 你

答复 全部答复 转发 ...
周三 2024/7/17 9:00

你好，我在GitHub上看到您发布的金铲铲之战项目，我想体验一下这个游戏项目，请问该项目如何启动？还有哪些运行环境需要设置？



Lin Minmus
收件人:

答复 全部答复 转发 ...
周四 2024/7/18 22:24

你好！

本项目开发引擎为：Cocos2d-x 3.17.2，在Windows平台下搭建开发环境可以参考Cocos官方文档：<https://docs.cocos.com/cocos2d-x/manual/zh/installation/Windows.html>

本项目使用的集成开发环境为：Microsoft Visual Studio 2022

编译游戏端可以通过VS2022打开proj.win32文件夹中的Teamfight_Tactic.sln启动，编译环境为Win32，想体验联机功能可以编译解决方案中的Server项目

您也可以直接从GitHub页面下载Release体验游戏

本项目仍有很多不足之处，欢迎提出Issues和PRs！

祝好！

答复 转发



Table of Contents

7 网络游戏状态同步

- ▶ Cocos2d-x 简介
- ▶ Cocos2d-x 基本概念
- ▶ Cocos2d-x 基本功能
- ▶ Cocos2d-x UI 组件
- ▶ Cocos2d-x 进阶内容
- ▶ Cocos2d-x 项目：Teamfight Tactics
- ▶ 网络游戏状态同步



网络游戏状态同步

Online Game State Synchronization

- 状态同步 (State Synchronization)
- 帧同步 (Frame Synchronization)



同濟大學
TONGJI UNIVERSITY



状态同步 (State Synchronization)

7 网络游戏状态同步

状态同步是指服务器定期向所有客户端发送游戏状态的完整数据，客户端通过接收这些数据来更新本地游戏的状态。每个客户端都保持独立的本地游戏状态，但定期与服务器同步，确保与其他客户端的状态一致。

工作原理：

- 服务器定期（例如每秒多次）将游戏的“全局状态”发送给所有连接的客户端。这个状态可以包括角色的位置、血量、战斗数据、技能冷却时间等信息。
- 客户端接收服务器的数据后，更新本地状态，避免因为网络延迟或丢包导致的状态不同步问题。
- 如果客户端在本地做了某些操作（如攻击），它会将这些操作发送给服务器，服务器会验证并处理这些操作，然后再将更新后的状态广播给所有客户端。



状态同步 (State Synchronization)

7 网络游戏状态同步

优势：

- 由于每次都同步完整的游戏状态，避免了较为复杂的逻辑处理。
- 客户端实现较为简单，不需要承担过多的计算和同步逻辑，减少了对客户端硬件的依赖。

缺点：

- 网络延迟和带宽占用较高，因为需要频繁传输大量的状态数据。
- 在快节奏的游戏（如战斗类游戏）中，延迟可能导致明显的卡顿或状态不一致。



状态同步 (State Synchronization)

7 网络游戏状态同步

战斗逻辑通常在服务器端实现。服务器会处理玩家的攻击、伤害判定、技能效果、玩家状态变化等逻辑，确保所有玩家看到的结果一致。客户端仅仅是负责显示战斗结果和响应用户的输入。



帧同步 (Frame Synchronization)

7 网络游戏状态同步

帧同步是一种基于时间帧的同步方式。游戏中的每个帧代表一个单位时间，所有玩家的客户端和服务器必须在相同的时间步（即“帧”）上进行同步，确保所有玩家在相同的时间点上进行相同的计算。

工作原理：

- 游戏会将每个游戏的“计算单位”分成多个“帧”（例如每秒 60 帧），每帧都包含一些计算逻辑（如玩家动作、战斗结果等）。
- 每个玩家的客户端和服务器都保持一个时间同步的机制，所有客户端都按照帧的顺序执行逻辑，并且在一帧的结束时同步彼此的游戏状态。
- 当玩家发起某个动作（如移动或攻击），这个动作会被客户端发送到服务器，服务器再根据该动作计算更新，并广播给所有玩家。
- 所有玩家的客户端都按照相同的时间步（帧）进行模拟，从而实现帧之间的一致性。



帧同步 (Frame Synchronization)

7 网络游戏状态同步

优势：

- 帧同步的游戏体验较为流畅，特别适合实时动作类游戏，因为它保证了每一帧的数据一致性。
- 适合低延迟环境，玩家之间的操作能够快速反应，减少卡顿感。

缺点：

- 帧同步对网络延迟非常敏感，任何一方的延迟都会影响到游戏体验。
- 游戏逻辑实现较复杂，客户端和服务器都需要严格同步帧数据。



帧同步 (Frame Synchronization)

7 网络游戏状态同步

帧同步中的战斗逻辑通常也在客户端和服务器端共同执行。客户端执行本地模拟，用户输入（如攻击）会根据时间步来传输给服务器，服务器对这些输入进行验证并决定战斗结果，然后再将结果同步给所有客户端。客户端按相同帧进行显示，确保所有玩家看到的战斗过程一致。



从零开始的 Cocos2d-x 开发指南

Starting from Scratch: A Cocos2d-x Development Guide

Thank you for your listening!

林继申 2024 年 12 月 1 日