字符串，标签和菜单

1. 字符串
2. c与c++的const char\* 和std::string
3. cocos2d::\_String 字符串
4. 定义：\_String \*name=\_String::create(“Hi Tony”)

\_String \*ns =\_String::createWithFormat(“%d”,num);//使用列表定义

（2）将\_String转换为const char \*: const char\*cstring=name->getCString();

其余的类型准换（其他转为\_String只需调用上述的构造函数）

（3）转换为int : int num=num->intValue();

二、标签

静态文字可以使用PS制作，不能被程序改变，动态文字可以用程序改变

主要标签类有：LabelTTF、LabelAtlas、LabelBMFont

1. LabelTTF
2. 说明：LabelTTF使用系统中的字体，是最简单的标签，继承了Node类，还实现了LabelProtocol接口
3. 实现的HelloWorld文字主要代码如下：//一般新建的项目都会把这一段设置//好，所以只需稍作修改即可,具体模板见下：

bool HelloWorld::init()

{

if(!scnen::init())

{return false;}……

//创建并初始化标签

auto label=LabelTTF::create(“Hello World”,” fonts/Marker Felt.ttf”,64);

label->setPoisition(Vec2(origin.x+visibleSize.width/2,origin.y+visibleSize.height-label->getContentSize().height));

this->addChild(lable,1);

…..

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

说明：create函数的第一个参数是要显示的文字，第二个参数是系统字体名，第三个参数是字体的大小（实际上create共有6个参数）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. LabelAtlas

图片集标签，这种标签显示的文字是从一个图片上集中取出的，这其中的每一个字符必须有固定的高度和宽度。

（1）间接继承Node,直接继承AtlasNode 还新增了LabelAtlas实现

（2）实现的HelloWorld文字主要代码如下：//一般新建的项目都会把这一段设//置好，所以只需稍作修改即可,具体模板见下：

bool HelloWorld::init()

{

if(!scnen::init())

{return false;}……

//创建并初始化标签

Auto label =LabelAtlas::create(“HelloWorld”,”fonts/tuffy\_bold\_italic-charmap.png0”,48,66,’ ’);

label->setPoisition(Vec2(origin.x+visibleSize.width/2- label->getContentSize().width/2,origin.y+visibleSize.height-label->getContentSize().height));

this->addChild(lable,1);

…..

return true;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

程序说明：第一个参数是要显示的文字，第二个参数是图片集文件，第三个参数是字符高度，第四个参数是字符宽度，第五个参数是开始字符

特别注意：使用LabelAtlas需要注意的是：图片必须放在Resources目录下；

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. LabelBMFont

(1)是位图字体标签，需要添加的文件一个图片集（.png）和一个字符坐标文件（.fnt）其中每一个字符的宽度是可变的。继承Node类，实现了LabelProtocol接口

(2)代码实现

bool HelloWorld::init()

{

if(!scnen::init())

{return false;}……

//创建并初始化标签

autolabel=LabelBMFont::create(“HelloWorld”,”fonts/BMFont.fnt”);

label->setPoisition(Vec2(origin.x+visibleSize.width/2,origin.y+visibleSize.height-label->getContentSize().height));

this->addChild(lable,1);

…..

return true;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

说明：第一个参数是要显示的对象，第二个参数是图片集文件。

注意：图片集文件和坐标文件需要放置在Resources目录下，文件名相同

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Label（最常用）

使用FreeType来使它在不同平台上有相同的视觉效果

1. 常用的create函数

①static Label\* createWithSystemFont(//创建系统字体标签对象

const std::string &text,//要显示的文字

const std::string&font,//系统字体名

float fontSize,//字体的大小

constSize& dimensions=Size::ZERO,//在屏幕上占用的区域的大小

TextHalignment hAlignment=TextHALignment::LEFT,//文字横向对齐方式，可省略

TextVALignment vAlignment=TextVALignment::TOP)//文字纵向对齐方式，可省略

②static Label\* createWithTTF (//创建TTF字体标签对象

const std::string &text,//要显示的文字

const std::string&fontFile,//字体文件

float fontSize,//字体的大小

constSize& dimensions=Size::ZERO,//在屏幕上占用的区域的大小，可省略

TextHalignment hAlignment=TextHALignment::LEFT,//文字横向对齐方式，可省略

TextVALignment vAlignment=TextVALignment::TOP)//文字纵向对齐方式，可省略

③ static Label\* createWithTTF (//创建TTF字体标签对象

const TTFConfig&ttfConfig,

const std::string&text,

TextHalignment hAlignment=TextHALignment::LEFT,

Int maxLinwWidth=0);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

说明：

1.TTFConfig是一个结构体变量

如：TTFConfig ttfConfig("fonts/Marker Felt.ttf", 36, GlyphCollection::DYNAMIC)

{

\_ttfConfig(const char\*filePath=””,//字体文件路径

int size=12,//字体大小

const GlyphCollection&glyphCollection=GlyphCollection::DYNAMIC,//字体库类型

const char\*customGlyphCollection=nullptr,//自定义字体库

bool useDistanceField=false,//开启距离字段字体开关

int outline=0//字体描边

}

2.ttfConfig.outlineSize=4;//设置TTFConfig的描边字体，字体描边

3.label->enableShadow(Color4B(255,255,255,128),Size(4,-4))//设置阴影的效果

4.label->setColor(Color3B::RED)//设置标签的颜色

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

④.static Label\* createWithBMfont (//创建位图字体标签对象

const std::string&bmfontFilePath,

const std::string &text,

const TextHalignment hAlignment=TextHALignment::LEFT,//文字横向对齐方式，可省略

int maxLinwWidth=0,

const Vec2& imageOffset=Vec2::ZERO)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

创建字体标签的步骤（下面的步骤均在Hello World.cpp的init函数中执行）

1. 确定需要建立的label类型

如：auto label = Label::createWithTTF();

1. 确定label类型的函数参数

如：auto label = Label::createWithTTF("Hello world2", "fonts/Marker Felt.ttf", 36);

auto label = Label::createWithTTF(ttfConfig,"Hello World3");

1. 对特殊情况进行一些修改

（1）auto label= Label::createWithTTF(ttfConfig,"Hello World3");

中需要在建立label标签之前创建结构体TTFConfig

1. 设计颜色、阴影等等的一些情况
2. 设置字体的位置

label->setPosition(Vec2(origin.x+visibleSize.width/2,origin.y + visibleSize.height - 200));

5.最后调用this->addChild(label, 1);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

注(1)VS中使用标签显示文字的时候有可能会出现乱码或者无法实现

(2)addChild()函数

函数原型：virtual void addChild(CCNode \* child, int zOrder, int tag);

第一个参数是要加入场景的指针

第二个参数是需要加入的层（数值越小最先放入，所以数值越大的再最上方）

第三个参数是给参数打上标签，类似于setTag,最终可以通过getChildTag找到

(3) getVisibleSize：获得视口（可视区域）的大小

getVisibleOrigin：逻辑区域的大小，比如为了适应屏幕时会等比例缩小或者放大，这个时候就可能在屏幕上出现

部分区域是空的（黑的），而此时的游戏画面的大小就是这个getVisibleOrigin。

getContentSize ： 来获得节点原始的大小。只是逻辑尺寸，不是像素

如label1->setPosition(Vec2(origin.x + visibleSize.width / 2, origin.y + visibleSize.height - 50));可以给label设定坐标

(4)Vec2是一个二维坐标类，其有两个参数，分别代表着横坐标(x)和纵坐标(y)

(5)给标签上色的两个方法：label->setColor(Color3B::YELLOW)//修改YELLOW处的宏解决或者label->setColor(ccc3(218,112,214));使用RGB染色。

RGB(255,192,203) pink（粉红）

■RGB(220,20,60) crimson（腥红）

■RGB(255,240,245) lavenderblush（紫罗兰红）

■RGB(219,112,147) palevioletred（淡紫红）

■RGB(255,105,180) hotpink（深粉红）

■RGB(199,21,133) mediumvioletred（适中的紫罗兰红）

■RGB(218,112,214) orchid（兰花紫）

■RGB(216,191,216) thistle（苍紫）

■RGB(221,160,221) plum（轻紫）

■RGB(238,130,238) violet（紫罗兰）

■RGB(255,0,255) magenta（洋紫)

■RGB(255,0,255) fuchsia（紫红）

■RGB(139,0,139) darkmagenta（深洋紫）

■RGB(128,0,128) purple（紫）

■RGB(186,85,211) mediumorchid（适中的兰花紫）

■RGB(148,0,211) darkviolet（深紫罗兰）

■RGB(75,0,130) indigo（靓青）

■RGB(138,43,226) blueviolet（蓝紫罗兰）

■RGB(147,112,219) mediumpurple（适中的紫）

■RGB(123,104,238) mediumslateblue（适中的的板岩蓝）

■RGB(106,90,205) slateblue（板岩蓝）

■RGB(72,61,139) darkslateblue（深板岩蓝）

■RGB(230,230,250) lavender（淡紫）

■RGB(248,248,255) ghostwhite（轻白）

■RGB(0,0,255) blue（蓝）

■RGB(0,0,205) mediumblue（适中的蓝）

■RGB(25,25,112)midnightblue（暗蓝）

■RGB(0,0,139) darkblue（深蓝）

■RGB(0,0,128) navy（海蓝）

■RGB(65,105,225) royalblue（皇家蓝）

■RGB(100,149,237) cornflowerblue（矢车菊蓝）

■RGB(176,196,222) lightsteelblue（淡钢蓝）

■RGB(119,136,153) lightslategray（浅石板灰）

■RGB(112,128,144) slategray（石板灰）

■RGB(30,144,255) dodgerblue（道奇蓝）

■RGB(240,248,255) aliceblue（爱丽丝蓝）

■RGB(70,130,180) steelblue（钢蓝）

■RGB(135,206,250)lightskyblue（淡天蓝）

■RGB(135,206,235) skyblue（天蓝）

■RGB(0,191,255) deepskyblue（深天蓝）

■RGB(173,216,230) lightblue（淡蓝）

■RGB(176,224,230)powderblue（火药蓝）

■RGB(95,158,160) cadetblue（军校蓝）

■RGB(240,255,255) azure（蔚蓝）

■RGB(224,255,255) lightcyan（淡青）

■RGB(175,238,238) paleturquoise（苍白的宝石绿）

■RGB(0,255,255) cyan（青）

■RGB(0,255,255) aqua（水绿）

■RGB(0,206,209) darkturquoise（深宝石绿）

■RGB(47,79,79) darkslategray（深石板灰）

■RGB(0,139,139) darkcyan（深青色）

■RGB(0,128,128) teal（水鸭色）

■RGB(72,209,204) mediumturquoise（适中的宝石绿）

■RGB(32,178,170) lightseagreen（浅海洋绿）

■RGB(64,224,208)turquoise（宝石绿）

■RGB(127,255,212) aquamarine（碧绿）

■RGB(102,205,170)mediumaquamarine（适中的碧绿）

■RGB(0,250,154) mediumspringgreen（适中的春天绿）

■RGB(245,255,250) mintcream（薄荷奶油）

■RGB(0,255,127) springgreen（春天绿）

■RGB(60,179,113)mediumseagreen（适中的海洋绿）

■RGB(46,139,87) seagreen（海洋绿）

■RGB(240,255,240)honeydew（浅粉红）

■RGB(144,238,144) lightgreen（浅绿）

■RGB(152,251,152) palegreen（苍白绿）

■RGB(143,188,143) darkseagreen（深海洋绿）

■RGB(50,205,50) limegreen（柠檬绿）

■RGB(0,255,0) lime（柠檬）

■RGB(34,139,34)forestgreen（森林绿）

■RGB(127,255,0) chartreuse（查特酒绿）

■RGB(124,252,0) lawngreen（草坪绿）

■RGB(173,255,47) greenyellow（绿黄）

■RGB(85,107,47) darkolivegreen（深橄榄绿）

■RGB(154,205,50) yellowgreen（黄绿）

■RGB(107,142,35) olivedrab（橄榄褐）

■RGB(245,245,220) beige（米色）

■RGB(250,250,210) lightgoldenrodyellow（浅秋黄）

■RGB(255,255,240) ivory（象牙白）

■RGB(255,255,224) lightyellow（浅黄）

■RGB(255,255,0) yellow（黄）

■RGB(128,128,0) olive（橄榄）

■RGB(189,183,107) darkkhaki（深卡其布）

■RGB(255,250,205) lemonchiffon（柠檬沙）

■RGB(238,232,170) palegoldenrod（灰秋）

■RGB(240,230,140) khaki（卡其布）

■RGB(255,215,0) gold（金）

■RGB(255,248,220) cornsilk（玉米）

■RGB(218,165,32)goldenrod（秋）

■RGB(184,134,11)darkgoldenrod（深秋）

■RGB(255,250,240)floralwhite（白花）

■RGB(253,245,230) oldlace（浅米色）

■RGB(245,222,179) wheat（小麦）

■RGB(255,228,181)moccasin（鹿皮）

■RGB(255,165,0) orange（橙）

■RGB(255,239,213)papayawhip（木瓜）

■RGB(255,235,205)blanchedalmond（漂白后的杏仁）

■RGB(255,222,173) navajowhite（耐而节白）

■RGB(250,235,215) antiquewhite（古白）

■RGB(210,180,140)tan（晒）

■RGB(222,184,135)burlywood（树干）

■RGB(255,228,196)bisque（乳脂）

■RGB(255,140,0) darkorange（深橙色）

■RGB(250,240,230) linen（亚麻）

■RGB(205,133,63) peru（秘鲁）

■RGB(244,164,96)sandybrown（沙棕）

■RGB(210,105,30) chocolate（巧克力）

■RGB(192,14,235) chocolatesaddlebrown（马鞍棕）

■RGB(255,245,238) seashell（海贝）

■RGB(160,82,45) sienna（土黄赭）

■RGB(255,160,122) lightsalmon（浅肉）

■RGB(255,127,80) coral（珊瑚）

■RGB(255,69,0) orangered（橙红）

■RGB(255,99,71)tomato（番茄色）

■RGB(255,228,225) mistyrose（雾中玫瑰）

■RGB(250,128,114)salmon（肉）

■RGB(255,250,250) snow（雪）

■RGB(240,128,128)lightcoral（浅珊瑚）

■RGB(188,143,143)rosybrown（玫瑰棕）

■RGB(205,92,92)indianred（浅粉红）

■RGB(255,0,0) red（红）

■RGB(165,42,42) brown（棕）

■RGB(178,34,34) firebrick（火砖）

■RGB(139,0,0) darkred（深红）

■RGB(128,0,0) maroon（粟色）

■RGB(255,255,255)white（白）

■RGB(245,245,245)whitesmoke（烟白）

■RGB(220,220,220)gainsboro（赶死部落）

■RGB(211,211,211) lightgrey（浅灰）

■RGB(192,192,192) silver（银白）

■RGB(169,169,169) darkgray（深灰）

■RGB(105,105,105) dimgray（暗灰）

■RGB(0,0,0) black（黑）

(6)Color 3B与Color4B

3B的参数是3个RGB参数，4B前3个是RGB，最后一个是透明度

三、菜单

菜单中又包含了菜单项，菜单项类是MenuItem,每个菜单项都有三个基本状态：正常、选中和禁止，它有MenuItemLabel文本菜单，MenuItemSprite精灵菜单,其有两个子类，MenuItemImage图片菜单；MenuItemToggle是开关菜单

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

建立背景图片：

1. 调用Sprite \*bg =Sprite::create(“menu/background.png”);//在menu目录下准备图片
2. bg->setPosition(Vec(origin.x+visibleSize.width/2,orign.y+visibleSize.height/2));

//设置放置图片的位置

（3）this->addChild(bg);//发布背景

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. 文本菜单（MenuItemLabel）

文本菜单只能显示文字

由于MenuItemLabel是抽象类，使用的时候需要调用它的两个子类MenuItemFont和MenuItemAtlasFont

MenuItemFont的create定义如下：

Static MenuItemFont \*create

(cons std::string &value,//要显示的文字

const ccMenuClaaback&callback)//菜单操作的回调函数指针

MenuItemAtlasFont的create函数定义如下：//是真的不会用

static MenuItemAtlasFont \*creat(cinst std::string&value，//要显示的文字

const std::string&charMapFile//图片集文件

int itemWidth,//要截取的文字在图片中的高度

int itemHeight,//要截取的文字在图片中的高度

char startCharMap,//开始字符

const ccMenuCallback &callback//菜单操作的回调函数指针

)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

写文本菜单的基本步骤：

1. 在每创建一个菜单之前可以使用MenuItem::setFontName()设置文本字体，用

MenuItemFont::setFontSize(int)来设置字体大小

1. 使用auto \*item=MenuItemFont::create()或auto \*item MenuItemAtlasFont::create()来创建文本菜单

如auto \*item=MenuItemFont::create(“Start”,CC\_CALLBACK\_1(HelloeWorld::menuItemCallback,this));可以创建一个文本菜单，CC\_CALLBACK\_1是一个回调函数，吧函数与对象绑定在一起，1表示这个函数有一个输出参数，

HelloWorld::menuItemCallback需要在HellowWorld.h头文件中声明（一般系统会自动生成一个），这一个Callback函数名可以自定义，但是要保证在类中声明并且添加定义

1. 在HelloWorld.h的头文件中的类声明中添加

Void menuItemCallback(cocos2d::Ref \*pSender);函数名要和之前的create函数的参数中的函数名相同。同时添加函数的定义。

Void HelloWorld::menuItemCallback(Ref \*pSender)

{

MenuItem \*item=(MenuItem \*)pSender;

log(“Touch Start Menu Item %p”,item);//其实回调函数里面可以随便写

}

1. 创建菜单对象，把之前创建的菜单项添加到菜单中

Menu \*mn=Menu::create(item,NULL);//最终一定要以NULL结尾。

1. 其他设置

如mn->alignItemsVertically()可以设置菜单垂直对齐

Item->setColor可以设置菜单的颜色

setPosition()来设置位置

1. 最终通过this->addChild(mn)发布；

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. 精灵菜单

菜单项类是MenuItemSprite，具有精灵的特点，可以让精灵动起来，具体操作是把一个精灵放置到菜单中作为菜单项。

精灵菜单类MenuItemSprite的创建函数

Static MenuItemSprite \*create(Node \*normalSprite,//菜单项正常显示时的精灵

Node \*selectedSprite,//选择菜单项时的精灵

Node \*disabledSprite,//菜单项禁用时的精灵

const ccMenuCallback &callback//菜单操作的回调函数指针

)

创建MenuItemSprite比较困难，因为要建立三个不同状态的精灵，有的create函数可以省略disabledSprite

大致操作：

Sprite\* startSpriteNormal = Sprite::create("menu/new-up.png");

Sprite\* startSpriteSelected = Sprite::create("menu/new-down.png");

MenuItemSprite\* startMenuItem = MenuItemSprite::create(startSpriteNormal, startSpriteSelected, CC\_CALLBACK\_1(HelloWorld::menuItemstartCallback, this));

startMenuItem->setPosition(Director::getInstance()->convertToGL(Vec2(100, 250)));

别忘了添加回调函数和addChild发布

1. 图片菜单

菜单项类是MenuItemImage,实际上图片菜单类继承于精灵菜单类

创建函数：

Static MenuItemImage \*create(const std::string&normalImage,//菜单项正常时的图片

const std::string&selsecedImage,//选择菜单时的图片

const std::string&disabledImage,//菜单禁用时的图片

const ccMenuCallback&callback,//菜单操作的回调函数指针

)

MenuItemImage有一些create函数，能够省略disabledImage参数

大致操作

MenuItemImage\* settingMenuItem = MenuItemImage::create("menu/setting-up.png", "menu/setting-down.png", CC\_CALLBACK\_1(HelloWorld::menuItemSettingCallback, this));

settingMenuItem->setPosition(Director::getInstance()->convertToGL(Vec2(200, 400)));

别忘了添加回调函数和addChild发布

1. 开关菜单

它是一种可以进行两种状态切换的菜单项

static MenuItemToggle \* create WithCallback(const ccMenuCallback &callback,//菜单操作的回调函数指针

MenuItem \*item//进行切换的菜单项，也可以是文本，图片和精灵类型的菜单项

…//最终以NULL结尾

)

例如auto toggleMenuItem=MenuItemToggle::createWithCallback(

CC\_CALLBACK\_(HelloWorld::menuItemCallback,this),

MenuItemFont::create(“On”),

MenuItemFont::create(“Off”),

NULL);