### 2.2 让人物动起来

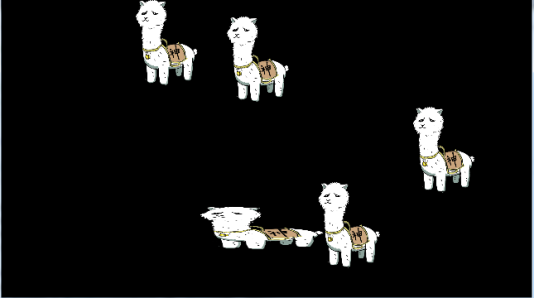
本节学习让游戏中的“角色”动起来的方法。这些方法包括让“角色”实现位置、大小的移动。还有实现人物动作的方法，包括使用动画来达到人物运动效果和对人物动画的帧进行打包的方法。

**【本节主要知识点】**

* 精灵进行简单的动作
* 曲线运动
* 动作队列的使用
* 利用动作监听实现卡牌翻转的动画效果
* 实现动画的方法
* 打包图片方便动画的创建

**（1）让精灵进行简单的动作**

本节通过示例学习在Cocos2d-x中实现简单运动的方法（水平、垂直，翻转运动）。初始状态5个图片整齐的排列，经过运动后，界面显示如下图所示。



代码：

// on "init" you need to initialize your instance

bool HelloWorld::init()

{

//////////////////////////////

// 1. super init first

if ( !Layer::init() )

{

return false;

}

Sprite\* m\_sprite[5]; **//创建5个精灵对象**

for (int i = 0; i < 5; i++) **//通过循环的方式将5个精灵对象加入到场景中**

{

m\_sprite[i] = Sprite::create("sprite.png");

m\_sprite[i]->setScale(0.2); **//大小**

m\_sprite[i]->setPosition(105 \* (i + 1), 80); **//位置**

addChild(m\_sprite[i]);

}

**//第1个精灵移动到坐标（200,300）位置，运动持续时间1.2秒**

auto\* moveto = MoveTo::create(1.2f,Vec2(200,300));

m\_sprite[0]->runAction(moveto);

**//第2个精灵相对原位置移动距离为（100,200），//第1个精灵移动到坐标（200,300）位置**

auto\* moveby = MoveBy::create(1.2f, Vec2(100, 200));

m\_sprite[1]->runAction(moveby);

**//第3个精灵分别在x，y两个方向进行缩放，x方向放大2倍，y方向为当前的0.5倍**

auto\* scaleby = ScaleBy::create(1.2f, 2.0f, 0.5f);

m\_sprite[2]->runAction(scaleby);

**//第4个精灵在原地闪烁4次，共1.2秒**

auto\* blink = Blink::create(1.2f, 4);

m\_sprite[3]->runAction(blink);

**//第5个精灵贝塞尔曲线运动**

ccBezierConfig bezier;

bezier.controlPoint\_1 = Vec2(-100,0);

bezier.controlPoint\_1 = Vec2(100, 80);

bezier.endPosition = Vec2(0,160);

auto\* bezierby = BezierBy::create(2.0f, bezier);

m\_sprite[4]->runAction(bezierby);

return true;

}

从示例中可以看出，实现精灵加单动作比较简单，就是先通过不同的类创建一个动作对象，然后再使精灵调用runAction方法实现。

上面示例实现中，每个精灵的运动，都不是一下子瞬时完成的，都有一个运动的过程。这类动作叫做延时动作。这些动作有一个共同的基类是IntervalAction，下表中列出了延时动作所使用的类及说明。

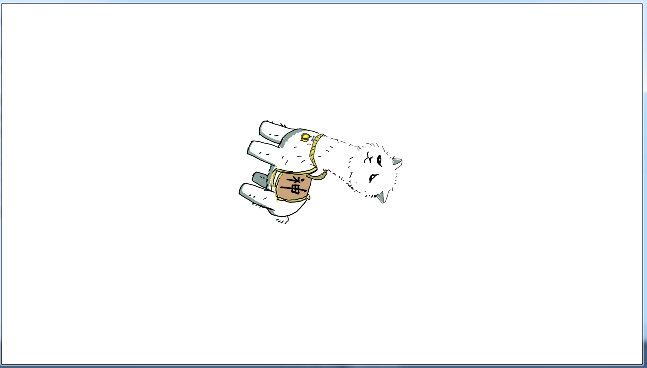
|  |  |
| --- | --- |
| **类名** | **说明** |
| MoveTo | 移动到某坐标 |
| MoveBy | 移动 |
| JumpTo | 跳跃到，需设置终点的位置和跳跃的位置和次数 |
| JumpBy | 跳跃，需设置终点位置和跳跃的高度和次数 |
| BezierTo | 贝塞尔曲线，支持3次贝塞尔曲线：P0-起点，P1-起点切线方向，P2-终点切线方向，P3-终点，采用绝对位移坐标，其中的参数是目标坐标 |
| BezierBy | 贝塞尔曲线，支持3次贝塞尔曲线：P0-起点，P1-起点切线方向，P2-终点切线方向，P3-终点，其中的参数是相对于初始坐标的偏移量 |
| ScaleTo | 放大到 |
| ScaleBy | 放大 |
| RotateTo | 旋转到 |
| RotateBy | 旋转 |
| Blink | 闪烁，需要设置闪烁的次数 |
| TintBy | 色调变化到 |
| FadeTo | 色调变化 |
| FadeIn | 由无变亮 |
| FadeOut | 由亮变无 |

**（2）复杂动作实现**

本节学习如何将简单动作组合在一起，实现复杂动作。

将动作组合在一起主要包括两个方面：连续动作和组合动作。如实现了精灵先从屏幕左下角运动到右上角，然后再折回到起点，最后又放大一倍的连续运动；在运动过程中同时又翻转的组合运动。

本节示例运行结果如下图所示。图中是精灵在运动过程中的截图。精灵初始在场景左侧，然后精灵将一边向右“滚动”，一边不断的膨胀，就是精灵进行移动和旋转所形成的效果。



**代码解释：**

**从下面的代码可以看出，只需将运动参数键入到spawn类的create方法中就可以了**

**Action\* action = Spawn::create(moveto,scaleto, rotateby, NULL);**

**RotateBy类的create方法，有两个参数，一个是动作执行的时间，另一个则是三维向量，表示精灵绕x，y，z轴旋转的角度。本例为仅绕z轴进行转动。**

**auto\* rotateby = RotateBy::create(2.5f, Vec3(0, 0, 240));**

实现代码：

// on "init" you need to initialize your instance

bool HelloWorld::init()

{

//////////////////////////////

// 1. super init first

if (!Layer::init() )

{

return false;

}

auto\* background = LayerColor::create(ccc4(255, 255, 255, 255));

addChild(background);

auto\* m\_sprite = Sprite::create("sprite.png");

m\_sprite->setPosition(100,180);

m\_sprite->setScale(0.2f);

addChild(m\_sprite);

auto\* moveto = MoveTo::create(2.5f,Vec2(540,200)); **//移动的动作**

auto\* scaleto = ScaleTo::create(2.5f,0.4); **//缩放的动作**

auto\* rotateby = RotateBy::create(2.5f, Vec3(0, 0, 240)); **//旋转的动作**

Action\* action = Spawn::create(moveto,scaleto, rotateby, NULL); **//同时执行3个动作**

m\_sprite->runAction(action); **//动作执行**

return true;

}

**（3）动作监听**

在实际使用中，还会需要改变动作的状态以达到连贯性的操作。如英雄向前冲刺，然后翻一个跟头，这就需要连续的组合运动。本节学习连续运动不同动作的功能。

本节示例运行后，界面效果如下图所示，分别显示了运行过程中的两个状态。程序运行后，在初始状态下一张卡牌显示在场景的中央，而当它转动到90度时，将会显示出另一面的图案。



**代码解释：**

**在场景中插入两张尺寸、位置均相同的卡片。其中一张为正面，直接方法场景中，而第二张卡片先旋转90度，因为这是2D游戏，卡片没有厚度，因此在用户的角度开来，第2张卡片是看不见的。**

**动作开始时，第1张卡片执行旋转90度的动作。随执行的是旋转90度，但是经过实际运行发现默认情况下，由于渲染的原因，其旋转的前半个圆周（视觉效果）实际上使用82.5为参数。**

**然后创建一个动作触发器，在被执行时触发其中的rotateCard方法。**

**Sequence类的create方法，可以包含多个参数，可以连续执行多个动作，最终以NULL作为最后一个参数结尾。**

**auto\* sequence = Sequence::create(rotateby, callFunc,NULL);**

**上面的方法是将两个动作加入到一个队列中，当执行完一个动过后，再执行下一个，只不过第2个动作是调用某个方法。看起来就很像事件监听**

代码：

bool HelloWorld::init()

{

//////////////////////////////

// 1. super init first

if ( !Layer::init() )

{

return false;

}

**//设置背景颜色为白色**

auto\* background = LayerColor::create(ccc4(255, 255, 255, 255));

addChild(background);

**//将两张卡牌插入到场景中，其中一张卡片使其旋转90度，达到看不见的状态**

myCard1 = Sprite::create("mycard01.png");

myCard2 = Sprite::create("mycard02.png");

myCard1->setPosition(360, 180);

myCard1->setScale(0.5f);

myCard1->setAnchorPoint(Vec2(0.5f, 0.5f));

addChild(myCard1);

myCard2->setPosition(360, 180);

myCard2->setScale(0.5f);

myCard2->setAnchorPoint(Vec2(0.5f, 0.5f));

**//以y轴为轴心旋转90度**

myCard2->setRotation3D(Vec3(0, 82.5f, 0));

addChild(myCard2);

**//以y轴为轴心旋转90度**

auto\* rotateby = RotateBy::create(1.5f, Vec3(0, 82.5f, 0));

**//创建动作监听**

auto\* callFunc = CallFunc::create(CC\_CALLBACK\_0(HelloWorld::rotateCard, this));

auto\* sequence = Sequence::create(rotateby, callFunc,NULL);

myCard1->runAction(sequence);

return true;

}

**//回调函数，第2张卡片以y轴为轴心旋转90度**

void HelloWorld::rotateCard()

{

auto\* rotateby = RotateBy::create(1.5f, Vec3(0, 97.5f, 0));

myCard2->runAction(rotateby);

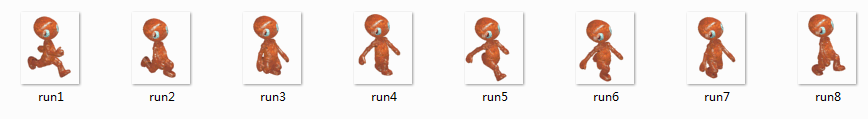
}

**（4）让角色跑起来**

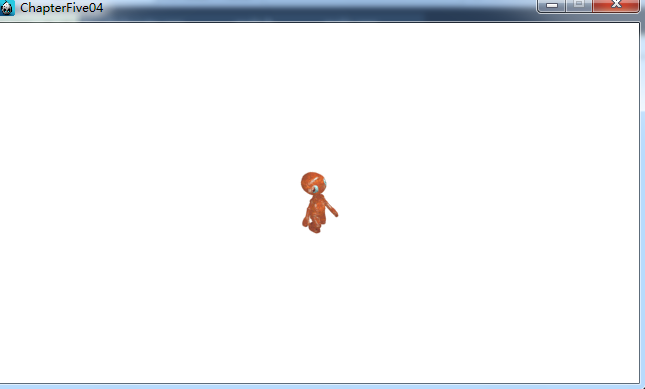
在游戏开发中还需要进行一些更加复杂的动作，比如人物的行走、跳跃等。虽然可以通过将人物分解为不同的部分，在进行组合，但这样无疑非常麻烦。

在Cocos2d-x中可以使用下面的方法实现：

首先准备一组连续的图片，并按照一定的规律命名，如下图。



在示例中，实现了如何让这8张图片连续地进行切换，以达到让人物行走的动作，执行结构如下所示。运行后，人物在原地做着行走的动作。



**代码解释：**

**首先要创建一个Animation类型的对象；**

**然后将素材中的图片通过循环一帧一帧地加入到里面；**

**将帧加入到Animation对象中；**

**使用create方法创建一个Animation对象，并执行。**

**设置动作执行的次数，-1表示无限制执行。**

**animation->setLoops(-1);**

**设置每一帧被切换的频率，即动作的播放速度，值越大，播放速度越快**

**animation->setDelayPerUnit(0.1f);**

代码：

bool HelloWorld::init()

{

//////////////////////////////

// 1. super init first

if ( !Layer::init() )

{

return false;

}

auto\* background = LayerColor::create(ccc4(255, 255, 255, 255));

addChild(background);

auto\* sprite = Sprite::create("run1.png");

sprite->setPosition(320, 180);

addChild(sprite);

auto\* animation = Animation::create();

for (int i = 1; i < 9; ++i)

{

char s[20] = { 0 };

sprintf(s, "run%d.png", i);

animation->addSpriteFrameWithFile(s);

}

animation->setLoops(-1); **//设置动作执行的次数，-1表示无限制执行**

animation->setDelayPerUnit(0.1f);**//设置每一帧被切换的频率，即动作的播放速度，值越大，播放速度越快**

auto\* action = Animate::create(animation);

sprite->runAction(action);

return true;

}

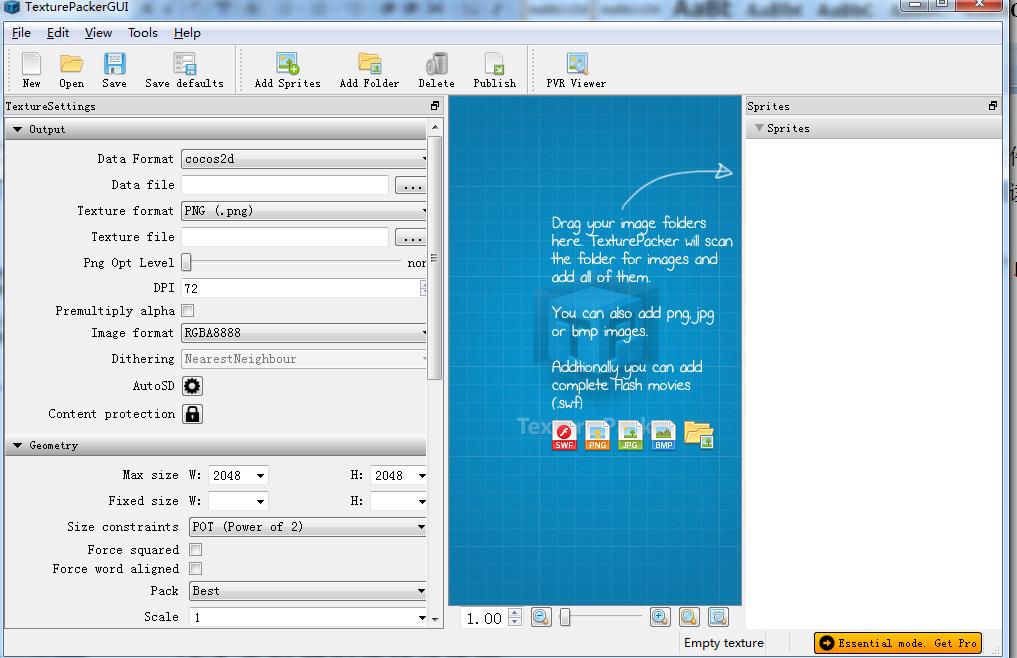
**（5）用打包后的图片创建动作**

前面方法利用一组连续的图像来播放动作，缺点是：浪费资源。上个示例用到8个素材图片，就需要连续读取8次图片，就造成了一定的资源浪费。如果一个完整的游戏所需要的素材无疑要更多，就需要一种办法来解决这个资源浪费问题。

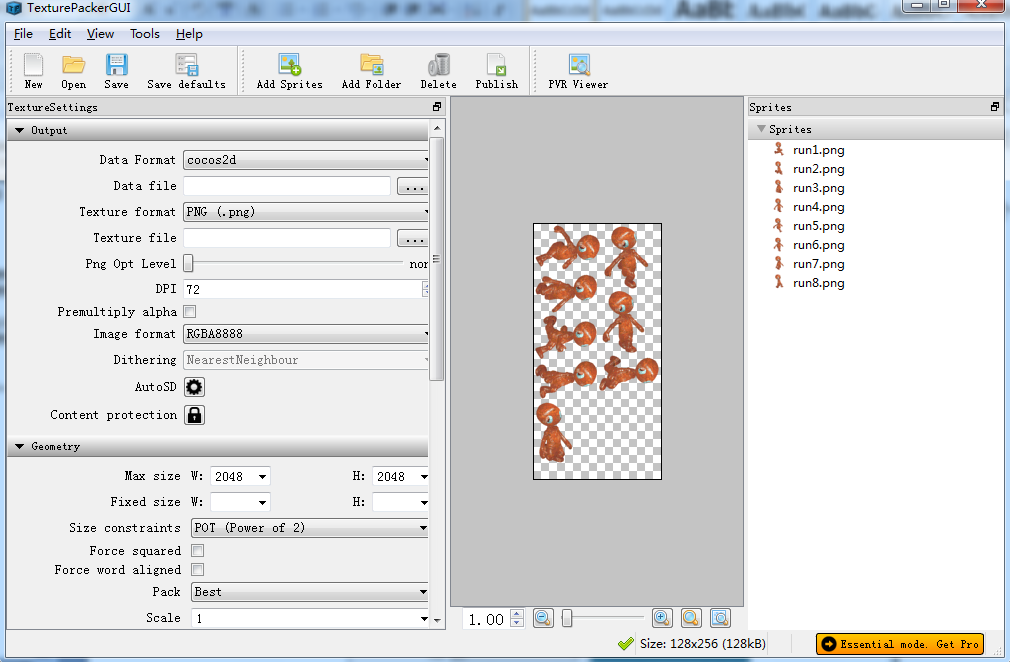
我们可以使用texturepacker软件来讲所需要的素材文件打包，获得一个大的图像文件和一个plist格式的XML文件，然后通过将它们读入到缓存中，达到只打开一次文件就实现读取全部素材的功能。

可以看到，texturepacker的作用就是将各种素材拼接在一张图片中，打包后得到一个plist文件，存放这张图片中的单位与原图片素材中的映射关系。

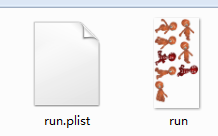
texturepacker安装后，运行界面如下图所示。单击工具栏中的“Add Sprites”按钮，导入需要的素材图片。



导入后，界面如下图所示：



然后单击Publish按钮保存生成的文件。在输出文件夹生成下图所示的两个文件。



下面的示例，通过使用run.png和run.plist文件实现任务行走动画。

**代码解释：**

**代码将动画文件加入缓存之后，通过原素材文件名调用它们。**

代码如下：

// on "init" you need to initialize your instance

bool HelloWorld::init()

{

//////////////////////////////

// 1. super init first

if ( !Layer::init() )

{

return false;

}

auto\* background = LayerColor::create(ccc4(255, 255, 255, 255));

addChild(background);

auto\* sprite = Sprite::create("run1.png");

sprite->setPosition(320, 180);

addChild(sprite);

auto\* m\_frameCache = SpriteFrameCache::getInstance();

m\_frameCache->addSpriteFramesWithFile("run.plist", "run.png");

Vector<SpriteFrame\*> frameArray;

for (int i = 1; i < 9; i++)

{

auto\* frame = m\_frameCache->getSpriteFrameByName(String::createWithFormat("run%d.png", i)->getCString());

frameArray.pushBack(frame);

}

Animation\* animation = Animation::createWithSpriteFrames(frameArray);

animation->setLoops(-1);//表示无限循环播放

animation->setDelayPerUnit(0.1f);//每两张图片的时间隔，图片数目越少，间隔最小就越小

//将动画包装成一个动作

auto\* action = Animate::create(animation);

sprite->runAction(action);

return true;

}

**【注意】使用该方法创建动作需要在使用完后清理缓存，清理缓存的方法如下：**

**AnimationCache::destroyInstance();**

**SpriteFrameCache::getInstance()->removeUnusedSpriteFrames();**

**TextureCache::getInstance()->removeUnuserdTextrues();**

**清理缓存的顺序为：先清理动画缓存再清理精灵帧缓存，最后才是清理缓存。**

**（6）实验2-3 位置移动（简单）**

**（7）实验2-4 位置移动（动作监听）**

**（8）实验2-5 简单人物走动**