

## 第七章 二维游戏动画合成

# 上节回顾

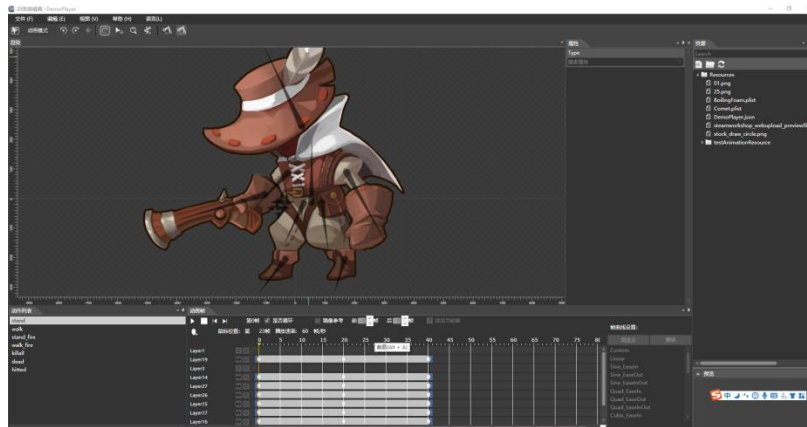
## • Chapter 7

### – Cocos2d-x动画编辑器

- 序列帧动画

- 骨骼动画

### – 游戏音效编程



# 本节内容

- Chapter 7

- Cocos2d-x中与动画相关的类

- 游戏动画实例----侠客行（上）

# 英雄快跑

- 英雄动画

- Animation

- 动画

- `auto animation=Animation::create();`

- 封装一系列动画帧

- `animation->addSpriteFrameWithFile( "player1.png" );`
- `animation->addSpriteFrameWithFile( "player2.png" );`

- 可设定动画帧的播放间隔

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

Chapter05 > player



player1.png



player2.png



player3.png



player4.png

# 英雄快跑

- 英雄动画
- Animate

Animation 只是存放了动画需要的数据信息，而执行动画还是需要 Animate

- 动画呈现
- 专门用来呈现动画的持续动作类

- auto animate=Animate::create(animation);
- sprite->runAction(animate);

# 英雄快跑

## • 英雄动画

```
25 // player img path
26 static string PLAYER_IMG_PATH[4] = {
27     "Chapter05/player/player1.png",
28     "Chapter05/player/player2.png",
29     "Chapter05/player/player3.png",
30     "Chapter05/player/player4.png"
31 } ;
```

```
cene.h  Config.h  MapScene.cpp  x
me  MapScene  update(float t)

169
170 void MapScene::addPlayer(Vec2 pos)
171 {
172     // 玩家跑动动画
173     Vector<SpriteFrame*> frameVector;
174     for(int i=0; i<4; i++)
175     {
176         auto spriteFrame = SpriteFrame::create(PLAYER_IMG_PATH[i], Rect(0, 0, PLAYER_WIDTH, PLAYER_HEIGHT));
177         frameVector.pushBack(spriteFrame);
178     }
179     auto animation = Animation::createWithSpriteFrames(frameVector);
180     animation->setDelayPerUnit(0.07f);
181     auto animate = Animate::create(animation);
182
183     // 添加玩家
184     auto player = Sprite::create();
185     player->setTag(PLAYER_TAG);
186     player->runAction(RepeatForever::create(animate));
187     this->addChild(player, 10);
188     player->setPosition(pos);
189 }
```

# 精灵

## 精灵的创建

可以使用一张图像来创建精灵, *PNG, JPEG, TIFF, WebP*, 这几个格式都可以。当然也有一些其它的方式可以创建精灵, 如使用 **图集** 创建, 通过 **精灵缓存** 创建, 我们会一个一个的讨论。本节介绍通过图像创建精灵。

## 使用图像创建

`Sprite` 能用一个特定的图像去创建:

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



上面直接使用了 `mysprite.png` 图像来创建精灵。精灵会使用整张图像, 图像是多少的分辨率, 创建出来的精灵就是多少的分辨率。比如图像是 200 x 200, `Sprite` 也是 200 x 200。

# 精灵

## 使用矩形

上一个例子，精灵和原始图像的尺寸一致。但是如果你想创建一个尺寸只有原始图像一部分的精灵，那你可以在创建的时候指定一个矩形，指定矩形需要四个值，初始 x 坐标，初始 y 坐标，矩形宽，矩形高。

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



矩形的初始坐标，从图形的左上角开始算，即左上角的坐标是 (0, 0)，不是从左下角。因此结果精灵是图像左上角的一小块，从左上角开始算起，40 x 40 的大小。

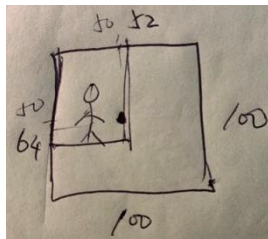


### 尺寸设置摘要

原始尺寸: 52 x 64 像素

调整后尺寸: 52 x 64 像素

```
static int PLAYER_WIDTH = 100;  
static int PLAYER_HEIGHT = 100;
```

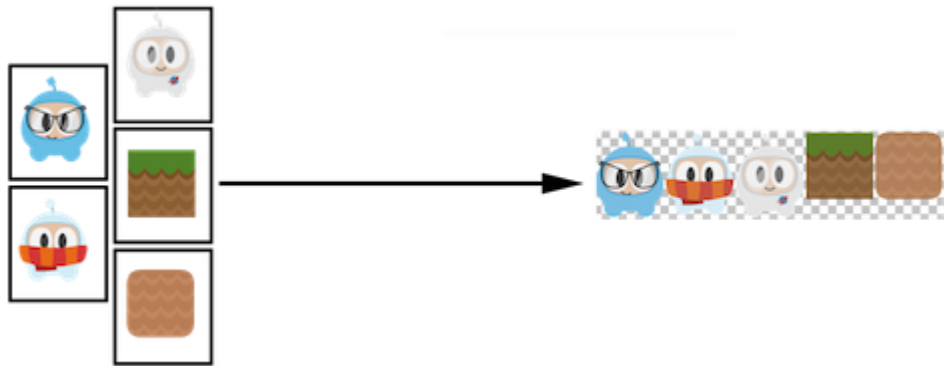


```
auto spriteFrame = SpriteFrame::create(PLAYER_IMG_PATH[i], Rect(0, 0, PLAYER_WIDTH, PLAYER_HEIGHT));  
frameVector.pushBack(spriteFrame);
```



## 使用图集

**图集(Sprite Sheet)** 是通过专门的工具将多张图片合并成一张大图，并通过 **plist 等格式的文件索引** 的资源，使用图集比使用多个独立图像占用的磁盘空间更少，还会有更好的性能。这种方式已经是游戏行业中提高游戏性能的标准方法之一。



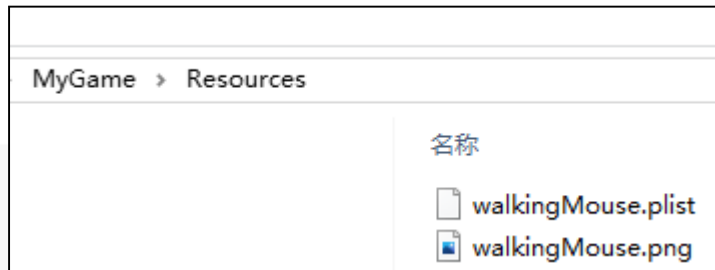
在使用图集时，首先将其全部加载到 `SpriteFrameCache` 中，`SpriteFrameCache` 是一个**全局的缓存类**，缓存了添加到其中的 `SpriteFrame` 对象，提高了精灵的访问速度。`SpriteFrame` 只加载一次，后续一直保存在 `SpriteFrameCache` 中。

## 加载图集

获取到 `SpriteFrameCache` 的实例，把图集添加到实例中。

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

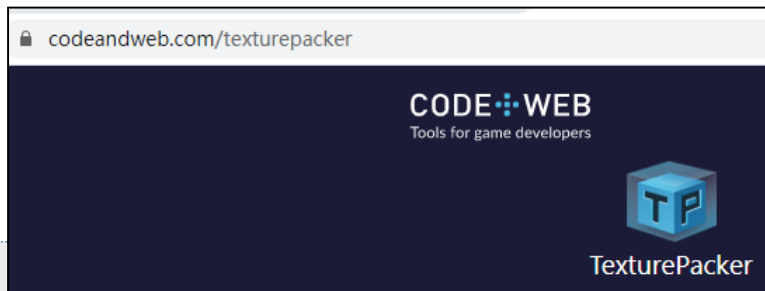
// the .plist file can be generated with any of the tools mentioned below
spritecache->addSpriteFramesWithFile("sprites.plist");
```



这样我们就完成了，将一个图集添加到 `SpriteFrameCache` 中，现在我们就利用这个对象创建精灵了！

## 创建图集

- [Texture Packer](#)



## 创建图集

- Texture Packer



## Tutorials for Cocos2d-X

20 seconds to your optimized  
sprite sheet

① Drop your sprites

② Choose your  
exporter

③ Press Publish

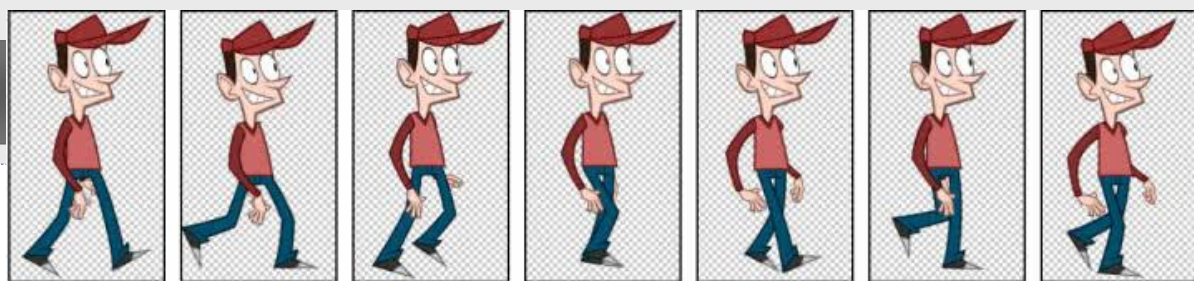
- ✓ save time
- ✓ save memory



**转场前**

# 精灵

## 使用精灵缓存



精灵缓存是 Cocos2d-x 为了提高精灵的访问速度，提供的一个精灵的缓存机制。

我们可以创建一个精灵并把精灵放到精灵的缓存对象 `SpriteFrameCache` 中：

```
// 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");
```

相对的，我们也可以从精灵的缓存对象 `SpriteFrameCache` 访问一个精灵，访问方法是先从缓存对象中获取对应的 `SpriteFrame`，然后从 `SpriteFrame` 创建精灵，方法：

```
// 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.png");
auto newSprite = Sprite::createWithSpriteFrame(newspriteFrame);
```

# 精灵

```
// load the Sprite Sheet
```

```
auto spritecache = SpriteFrameCache::getInstance();
```

```
// the .plist file can be generated with any of the tools mentioned below
```

```
spritecache->addSpriteFramesWithFile("walkingMouse.plist");
```

```
// walkingMouse animation demo
```

```
int num = 17; // number of pictures for walking mouse
```

```
auto animation = Animation::create();
```

```
auto sprite = Sprite::create();
```

```
for (int i = 1; i < num; i++)
```

```
{
```

```
    char name[20];
```

```
    if (i < 10)
```

```
    {
```

```
        sprintf(name, "0%d.png", i);
```

```
    }
```

```
    else
```

```
    {
```

```
        sprintf(name, "%d.png", i);
```

```
    }
```

```
    auto newspriteFrame = spritecache->getSpriteFrameByName(name);
```

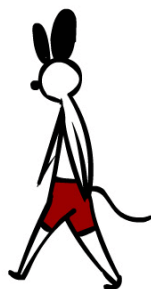
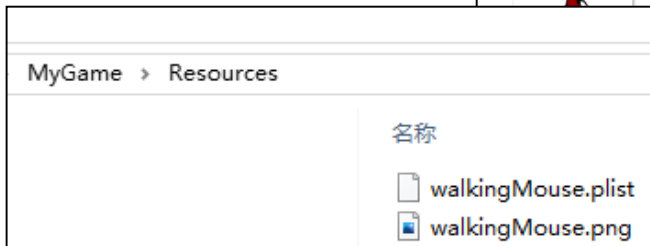
```
    animation->addSpriteFrame(newspriteFrame);
```

```
}
```

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

```
auto animate = Animate::create(animation);
```

```
sprite->runAction(RepeatForever::create(animate));
```



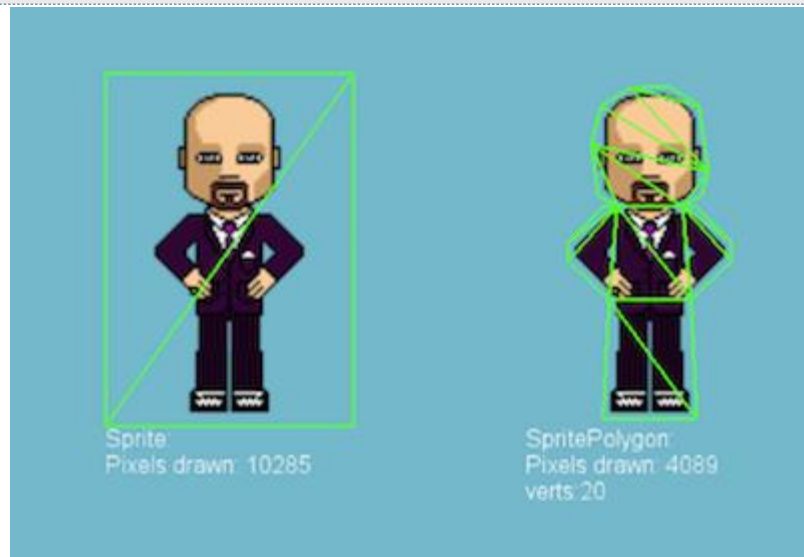
**转场前**

# 精灵

## 多边形精灵

### 为什么要使用多边形精灵

提高性能!



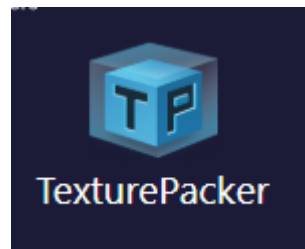
左侧，是一个典型的精灵绘制时的处理，精灵被处理成一个有两个三角形组成的矩形。

右侧，是一个多边形精灵绘制时的处理，精灵被处理成一系列小的三角形。

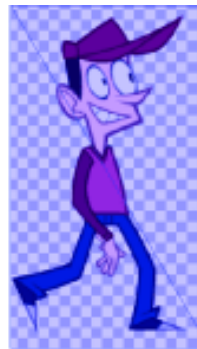
显然可以看到，右侧多边形精灵需要绘制的像素数量比左侧精灵需要的像素数量更小，但是由于划分了多个三角形出现了更多的顶点，由于在现代的图形处理中，一般绘制定点比绘制像素消耗的性能少，所以多边形精灵的性能更好，实际的测试结果也验证了这一点。



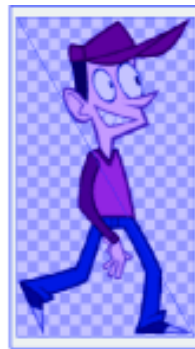
# 精灵



## AutoPolygon



Original  
100% overdraw



Rectangular Trim  
83% overdraw



Polygon Trim  
35% overdraw



**AutoPolygon** 是一个工具类，它可以在程序运行时，通过跟踪关键点和三角测量，将一个矩形图像划分成一系列小三角形块。

首先将图像资源传入 **AutoPolygon** 进行处理，然后我们使用它生成的对象进行精灵的创建就能得到多边形精灵。

```
// Generate polygon info automatically.  
auto pinfo = AutoPolygon::generatePolygon("filename.png");  
  
// Create a sprite with polygon info.  
auto sprite = Sprite::create(pinfo);
```

# 本节内容

- Chapter 7
  - Cocos2d-x中与动画相关的类
  - 游戏动画实例----侠客行（上）

# 游戏动画实例——侠客行

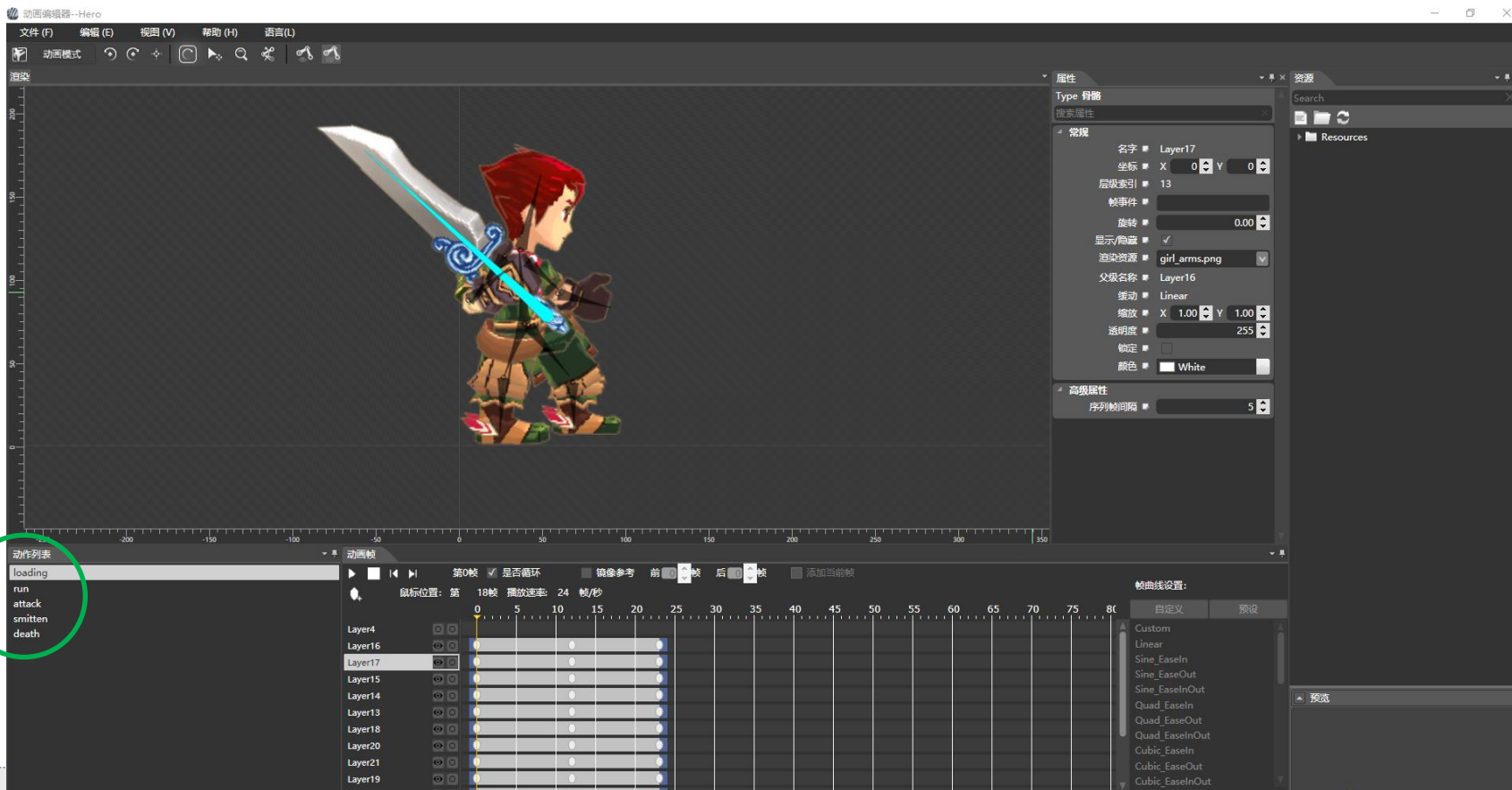


# 侠客行（教材demo）

**转场前**

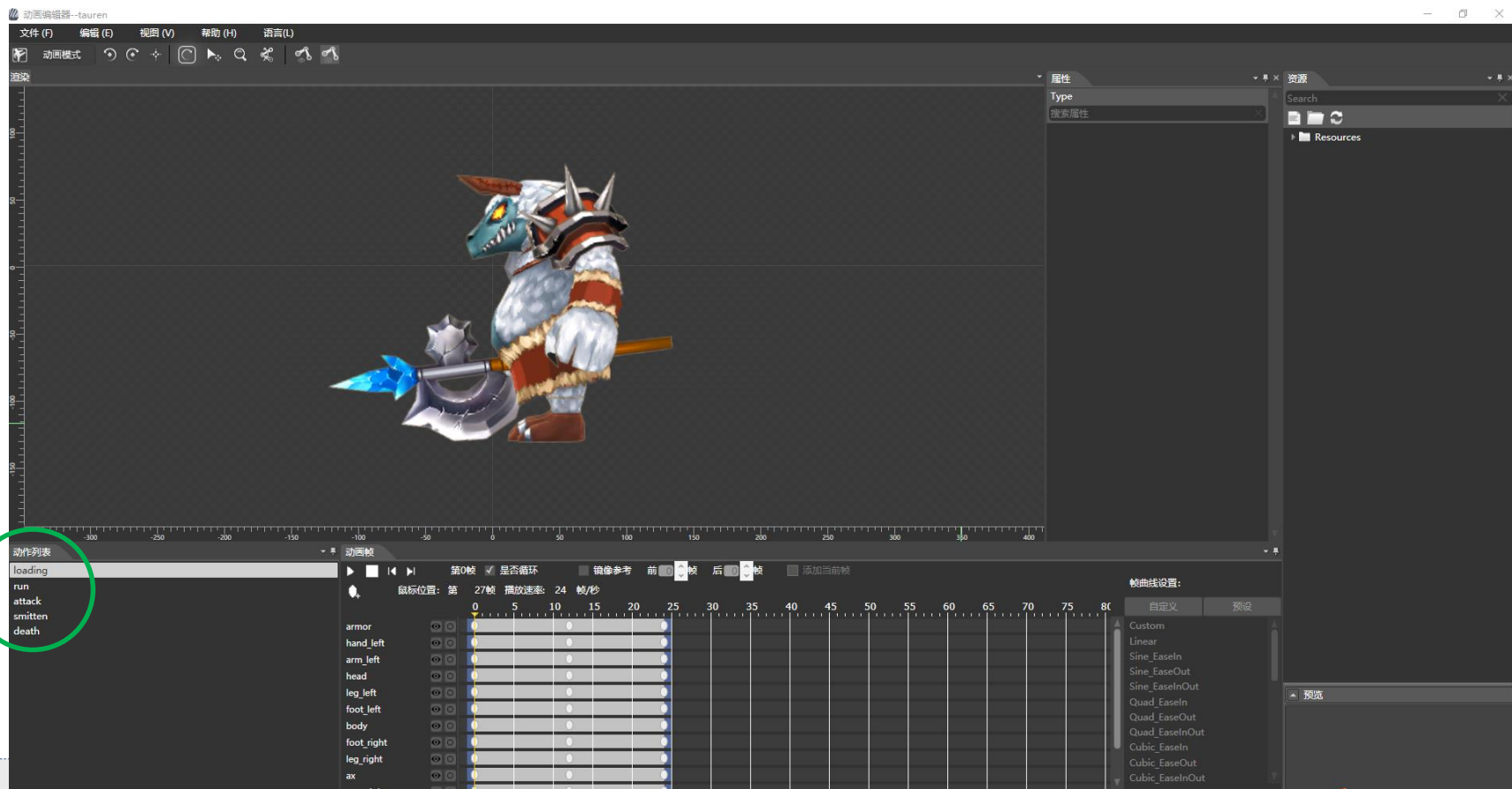
# 游戏动画实例——侠客行

- 本例使用的素材资源来源于cocos studio示例文件



# 游戏动画实例——侠客行

- 本例使用的素材资源来源于cocos studio示例文件

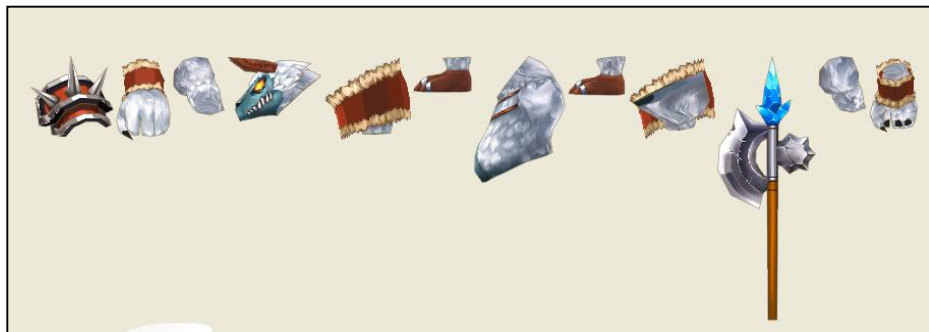


# 动画编辑器示例项目

**转场前**




# 骨骼动画调用

- 导出骨骼动画
  - 选择文件->导出项目
  - 使用默认参数导出动画



## 导出项目

导出完毕后，把导出的文件夹拷贝到cocos2d-x project的Resource文件夹下，就能够在项目中使用。

e > bin > mythirdproject > Game > <b>Resources</b> > Chapter06 > Editor > tauren			
名称	修改日期	类型	大小
 tauren.ExportJson	2015/11/2 20:04	EXPORTJSON 文...	190 KB
 tauren0.plist	2015/11/2 20:04	PLIST 文件	8 KB
 tauren0.png	2015/11/2 20:04	PNG 文件	146 KB



# 骨骼动画调用

头文件添加引用  
&  
声明成员变量

```
31 // start animation first time
32 bool m_startAnimation;
33
34 // animation
35 Armature *m_armature;
```

AnimationEditorScene.h AppDelegate.cpp main.cpp AnimationEditorScene.cpp

Game (全局范围)

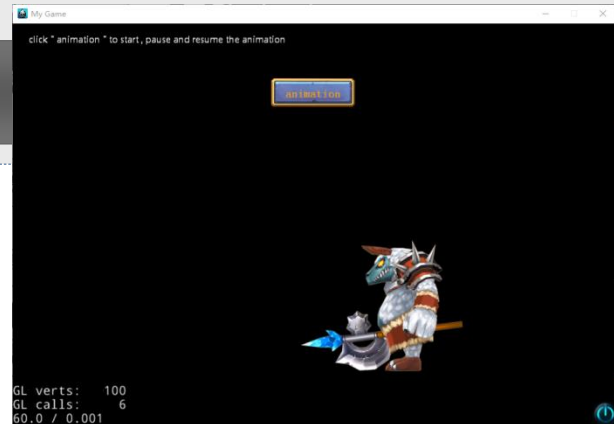
```
1 // 第六章例子1 -- 动画编辑器的应用
2
3 #ifndef __ANIMATION_EDITOR_SCENE_H__
4 #define __ANIMATION_EDITOR_SCENE_H__
5
6 // cocos2d
7 #include "cocos2d.h"
8 #include "cocostudio/CocoStudio.h"
9 #include "ui/CocosGUI.h"
10 using namespace cocos2d;
11 using namespace cocostudio;
12 using namespace cocos2d::ui;
```



# 骨骼动画调用

## 初始化函数中

bool AnimationEditorScene::init()



```
70     ArmatureDataManager::getInstance()->addArmatureFileInfo("Chapter06/Editor/tauren.ExportJson");
71     m_armature = Armature::create("tauren");
72     if (m_armature == NULL)
73     {
74         CCLOG("animation load failed!");
75         return false;
76     }
77     m_armature->setPosition(visibleSize.width/2 + 100, visibleSize.height/2 - 100);
78     this->addChild(m_armature);
```

```
// show ui
```

```
this->addChild(layout_root);
```

```
// add button click callback
```

```
auto btn_test_ani = (Button *)layout_root->getChildByTag(5);
```

```
btn_test_ani->addTouchEventListener(CC_CALLBACK_2(AnimationEditorScene::onClick, this));
```

# 骨骼动画调用

## 按钮回调函数

```
3 AnimationEditorScene::AnimationEditorScene()  
4 {  
5     m_startAnimation = false;  
6     m_armature = NULL;  
7 }
```

```
89 // button test callback  
90 void AnimationEditorScene::onClick(Ref *pSender, Widget::TouchEventType type)  
91 {  
92     switch (type)  
93     {  
94         break;  
95         case cocos2d::ui::Widget::TouchEventType::ENDED:  
96  
97             if (!m_startAnimation && (m_armature->getAnimation()->isPause() == true))  
98             {  
99                 //m_armature->getAnimation()->playWithIndex(0); // start animation  
100                 m_armature->getAnimation()->play("run");  
101                 //m_armature->getAnimation()->play("attack");  
102             } else if (m_armature->getAnimation()->isPause() == true)  
103             {  
104                 m_armature->getAnimation()->resume(); // resume animation  
105             } else  
106             {  
107                 m_armature->getAnimation()->pause(); // stop animation  
108             }  
109  
110             m_startAnimation = true;  
111             break;
```

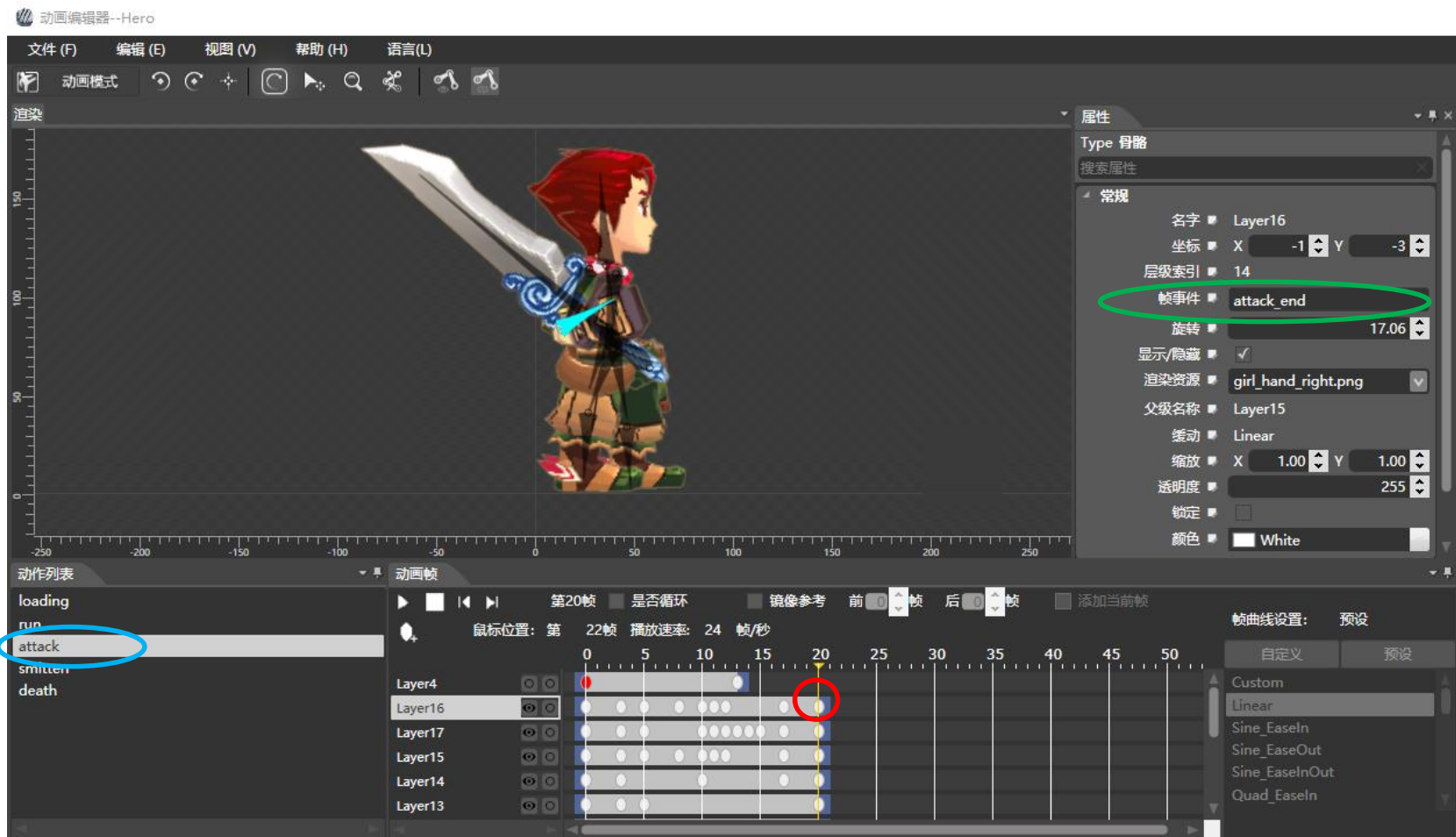


**转场前**

# 游戏动画实例——侠客行

- 动画编辑器示例项目中的人物现有动作
  - 可以不经修改就能满足实验的需求
  - 但是需要在一些动作的末尾添加帧事件
- 例如：
  - 1 在loading动作的最后一帧，选中该帧，在右侧属性栏的帧事件写入loading\_end
  - 2 相同方法在攻击动画最后一帧加上帧事件attack\_end
  - 3 .....

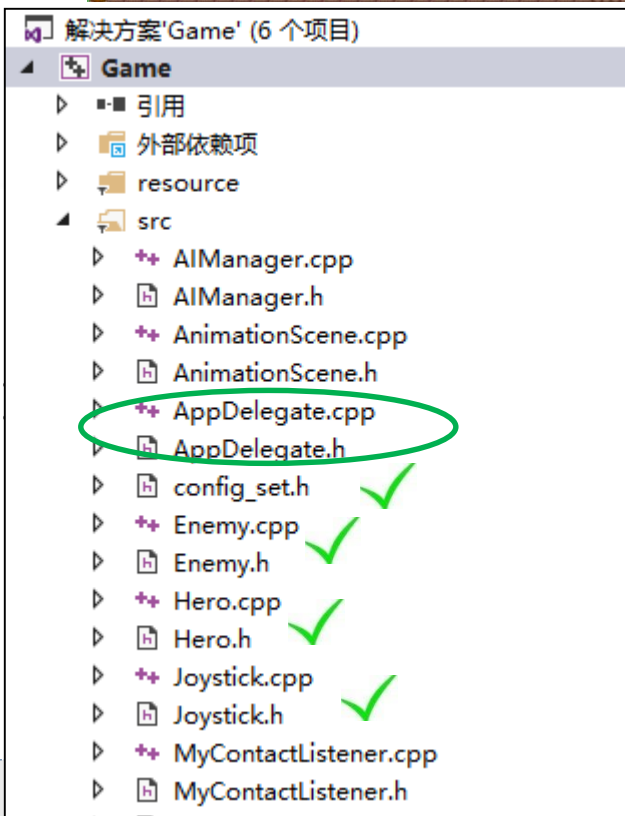
# 游戏动画实例——侠客行



# 游戏动画实例——侠客行

- 类的封装

- 头文件包含及常量的定义放在单独的文件`config_set.h`内
- 封装英雄类`Hero`
- 封装敌人类（和英雄类类似）`Enemy`
- 封装摇杆类`Joystick`



# 游戏动画实例——侠客行

- 类的封装

- 碰撞检测类

MyContactListener

- 统一管理游戏场景的碰撞检测

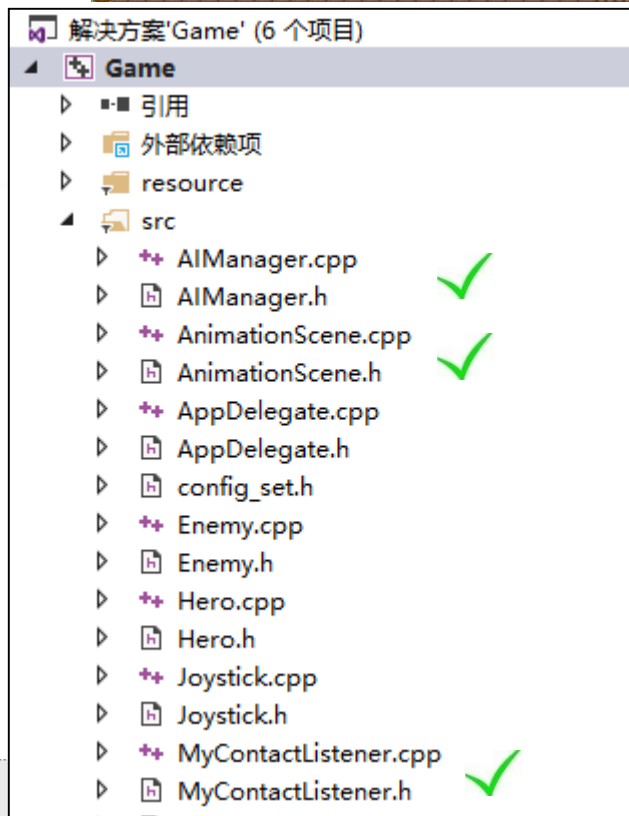
- AI管理类

AIManager

- 管理敌人的智能行为

- 主场景类

AnimationScene

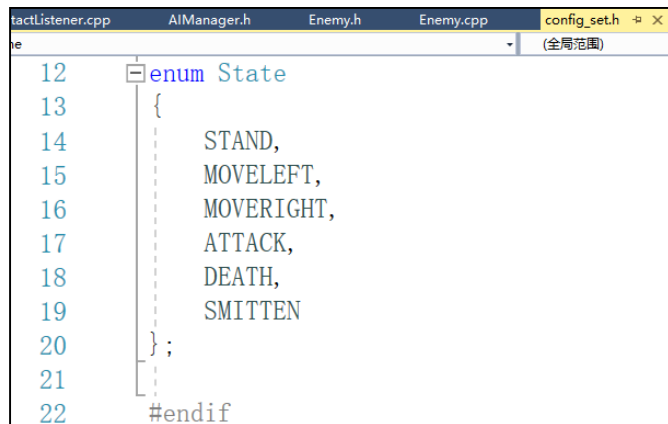




# 游戏动画实例——侠客行

## • 程序逻辑的实现

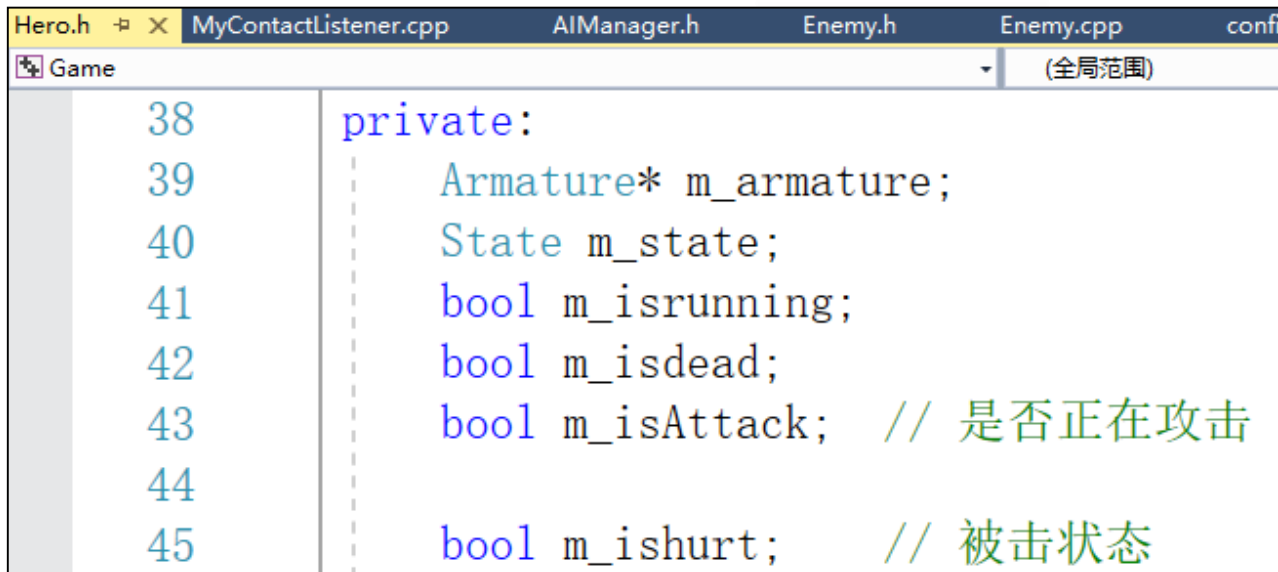
- 玩家和AI分别扮演英雄Hero和敌人Enemy
- 英雄和敌人的逻辑实际上是一样的，将会对玩家的操作反馈出站立、跑动（左右）、攻击、颤动、死亡等动画
- 这些动画播放的实现由一个状态量State来控制，包括 STAND、MOVE (LEFT、RIGHT)、ATTACK等



# 游戏动画实例——侠客行

- 程序逻辑的实现

- 为了保证动画的播放，防止因为停滞在某个状态而一直播放某个动画的第一帧等，英雄Hero和敌人Enemy还需要拥有一系列的布尔类型变量



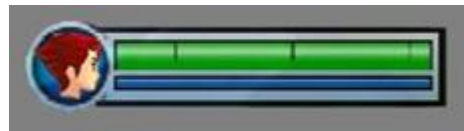
The screenshot shows a code editor with several tabs: Hero.h, MyContactListener.cpp (active), AIManager.h, Enemy.h, Enemy.cpp, and confi. The active file, MyContactListener.cpp, is in the 'Game' scope (全局范围). The code shows the private member variables of the Hero class, starting from line 38:

```
38 private:
39     Armature* m_armature;
40     State m_state;
41     bool m_isrunning;
42     bool m_isdead;
43     bool m_isAttack; // 是否正在攻击
44
45     bool m_ishurt; // 被击状态
```

# 游戏动画实例——侠客行



- Hero类
  - 继承自Sprite



```
class Hero : public Sprite
```

- 成员变量

- 布尔变量
- 骨骼动画变量
- 状态变量
- 生命值变量

```
38 private:
39     Armature* m_armature;
40     State m_state;
41     bool m_isrunning;
42     bool m_isdead;
43     bool m_isAttack;    // 是否正在攻击
44
45     bool m_ishurt;      // 被击状态
46
47     int m_life;         // 生命值
48
49     int m_max_life;     // 最大生命值
```

# 游戏动画实例——侠客行

- Hero类

- 成员方法

- 用于实例化和更新状态的声明

```
8      public:  
9          Hero();  
10         ~Hero();  
11  
12         static Hero* create(Vec2 position);  
13  
14         void update(float delta);
```

- 用于初始化和骨骼动画回调的声明

```
22     virtual bool init(Vec2 position);  
23     void onFrameEvent(cocostudio::Bone *bone, const std::string& evt, int originFrameIndex, int currentFrameIndex);
```



# 游戏动画实例——侠客行

- Hero类

- 成员方法

- 提供给外部调用的方法，如：设置当前状态量为应该播放的动画状态（攻击、被攻击或防御等）

```
16      void play(State state);
```

- 供外部（如碰撞检测组件）调用的函数，表示英雄被攻击，效果是产生伤害并显示数值

```
17      void hurt(); // 被击中
18
19      void showBloodTips(int s); // 显示扣血数字
20      void flyend(Label* label); // 扣血数字移动并消失
```

# 游戏动画实例——侠客行

- Hero类

- 成员方法

- 获取/设置成员变量的方法，用以外部获取或者设置hero的行为信息

```
25         // set and get
26         Armature* getArmature() { return m_armature; }
27
28         bool isAttack() { return m_isAttack; }
29         void setAttack(bool attack) { m_isAttack = attack; }
30
31         int getLife() { return m_life; }
32         void setLife(int life) { m_life = life; }
33
34         bool isDeath() { return m_isdead; }
35
36         int getMaxLife() { return m_max_life; }
```

# 游戏动画实例——侠客行

- Hero类的实现

- 构造函数

- 初始化成员变量



```
3 Hero::Hero()  
4 {  
5     m_isrunning = false;  
6     m_isdead = false;  
7     m_isAttack = false;  
8     m_ishurt = false;  
9     m_max_life = 500;  
10    m_life = m_max_life;  
11 }
```

- 初始化函数

- Hero::init(Vec2 position)
    - 添加骨骼动画

```
40 ArmatureDataManager::getInstance()->addArmatureFileInfo("Chapter06/AnimationScene/animation/Hero/Hero.ExportJson");  
41 m_armature = Armature::create("Hero");  
42 if (m_armature == NULL)  
43 {  
44     CCLOG("hero load error!");  
45     return false;  
46 }
```

# 游戏动画实例——侠客行

- Hero类的实现

- 初始化函数

- Hero::init(Vec2 position)
    - 设置位置
    - 播放待机动画
    - 添加帧动画监听器
    - 启动调度器update



```
47 m_armature->setPosition(Vec2::ZERO);
48 m_armature->getAnimation()->play("loading");
49 m_armature->getAnimation()->setFrameEventCallFunc(CC_CALLBACK_0(Hero::onFrameEvent,
50 this->addChild(m_armature);
51 this->setPosition(position);
52
53 this->scheduleUpdate();
```



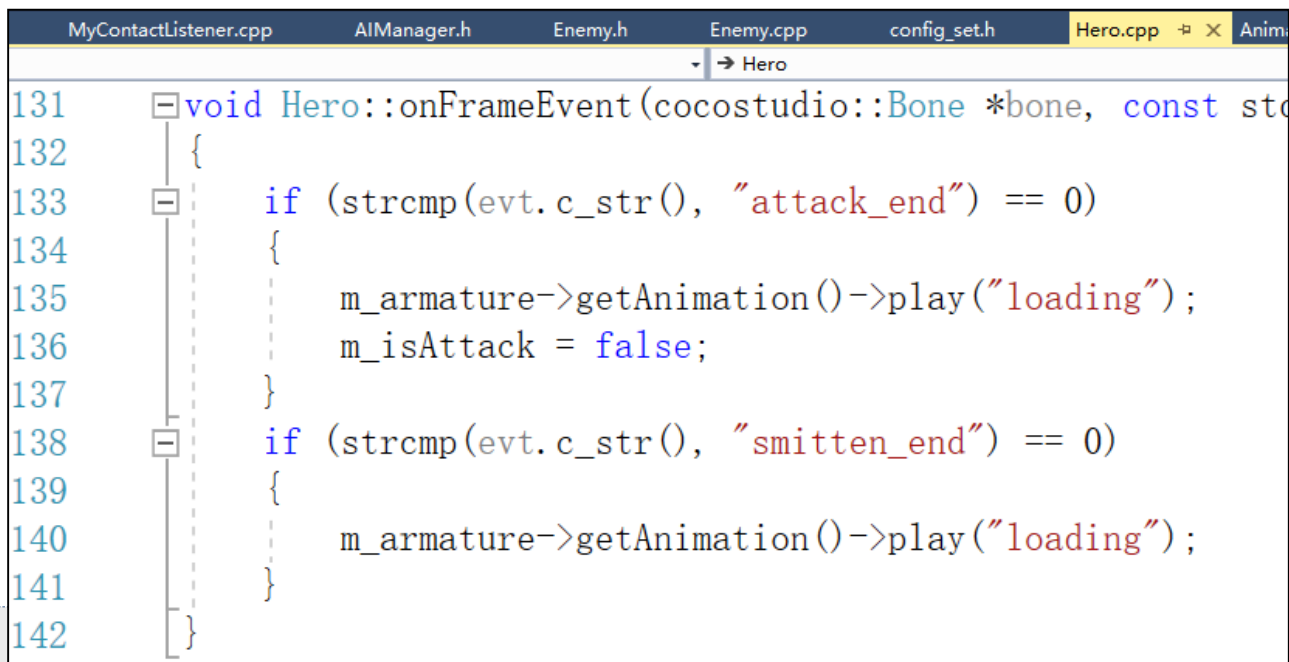
# 游戏动画实例——侠客行

## – 为骨骼添加帧动画监听器

- 监听特定动画结束帧

- 还原布尔类型变量，避免重复播放动画

- 让英雄回归正常待机状态



```
MyContactListener.cpp  AIManager.h  Enemy.h  Enemy.cpp  config_set.h  Hero.cpp  Anim.  
- -> Hero  
131 void Hero::onFrameEvent(cocostudio::Bone *bone, const std::string &evt) {  
132 {  
133     if (strcmp(evt.c_str(), "attack_end") == 0)  
134     {  
135         m_armature->getAnimation()->play("loading");  
136         m_isAttack = false;  
137     }  
138     if (strcmp(evt.c_str(), "smitten_end") == 0)  
139     {  
140         m_armature->getAnimation()->play("loading");  
141     }  
142 }
```

# 游戏动画实例——侠客行

## – Hero::update(float dt)

- 每帧**检测**当前的状态量m\_state
- 使用switch-case语句**切换**到对应的动画播放

```
58 void Hero::update(float delta)
59 {
60     if (m_life <= 0)
61     {
62         play(DEATH);
63     }
64     switch (m_state)
65     {
66     case STAND:
```



- 不能简单的直接播放动画。由于update()每帧都会执行一次，如果不加以控制，将会**持续播放**某个动画的**第一帧**。而这就是一系列bool类型状态量存在的意义了

# 游戏动画实例——侠客行

## – Hero::update(float dt)

- 以STAND为例



- 显然，死亡为true时就不能播放任何动画了（除死亡动画）
- 正在攻击、正在被攻击、正在跑动等时刻自然也不能播放
- 播放loading动画之后，需要把正在跑动设置为false，这是因为如果不这样设置，hero将不会响应下一次跑动指令

```
64  switch (m_state)
65  {
66  case STAND:
67      if ((m_isrunning == true) && (m_isdead == false) && (m_ishurt == false))
68      {
69          m_armature->getAnimation()->play("loading");
70          m_isrunning = false;
71      }
72      break;
```

# 游戏动画实例——侠客行

## – Hero::update(float dt)

- 以MOVELEFT为例
  - 边界判断
  - 方向判断 (水平翻转)
  - 位置更新

其它情况同理  
自行分析

```
73 case MOVELEFT:
74     if ((this->getPositionX() > 0) && (m_isdead == false) && (m_ishurt == false) && (m_isAttack == false))
75     {
76         if (m_isrunning == false)
77         {
78             m_armature->getAnimation()->play("run");
79             m_isrunning = true;
80         }
81         if (m_armature->getScaleX() != -1)
82         {
83             m_armature->setScaleX(-1);
84         }
85         this->setPositionX(this->getPositionX() - 4);
86     }
87     break;
```

# 游戏动画实例——侠客行

## – Hero::play(State state)

- 供外部调用：设置Hero的状态量m\_state

```
144 void Hero::play(State state)
145 {
146     if (state == SMITTEN) // 控制被击中时颤抖动画只播放一次
147     {
148         m_ishurt = true;
149     }
150     m_state = state;
151 }
```

## – Hero::hurt()

- 供外部调用：英雄被攻击后，产生伤害并显示数值

# 游戏动画实例——侠客行

– Hero::hurt()

```
153 void Hero::hurt()  
154 {  
155     // 根据基础伤害造成随机伤害  
156     showBloodTips(30);  
157     this->play(SMITTEN);  
158 }
```

– showBloodTips()

```
160 // 显示扣血  
161 void Hero::showBloodTips(int s)  
162 {  
163     int hitCount = 1;  
164     int hitRand = rand()%10;  
165     if (hitRand > 3 && hitRand < 8)  
166     {  
167         hitCount = 2;  
168     } else if (hitRand > 7)  
169     {  
170         hitCount = 3;  
171     }
```

# 游戏动画实例——侠客行

## – showBloodTips (int s)

```
173  for (int i = 0; i < hitCount; i ++)  
174  {  
175      int hurt_blood = s + rand()%8;  
176      setLife(m_life - hurt_blood); // 扣血  
177      auto label = Label::createWithBMFont("fonts/futura-48.fnt", StringUtils::format("%d", hurt_blood));  
178      label->setColor(Color3B::RED);  
179      this->addChild(label, 5);  
180      label->setPosition(Vec2(0, 0) + Vec2(20 + rand()%80, 10 + rand()%80));  
181      label->runAction(Sequence::create(  
182          MoveBy::create(0.7f, Vec2(0, 30)),  
183          CallFunc::create(CC_CALLBACK_0(Hero::flyend, this, label)),  
184          NULL  
185      ));  
186  }
```

## – flyend( Label\* label)

```
189  void Hero::flyend(Label* label)  
190  {  
191      label->setVisible(false);  
192      label->removeFromParent();  
193  }  
194
```