

## 第七章 二维游戏动画合成

# 上节回顾

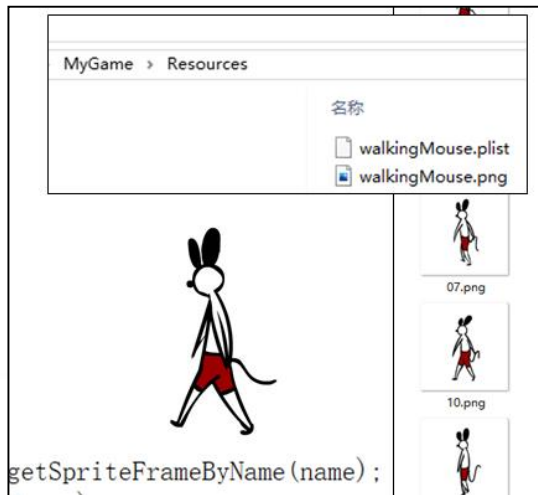
## • Chapter 7

### – Cocos2d-x动画编辑器

- 序列帧动画
- 骨骼动画

### – 提高计算机动画效果的基本手法

### – Cocos2d-x中与动画相关的类



# 本节内容

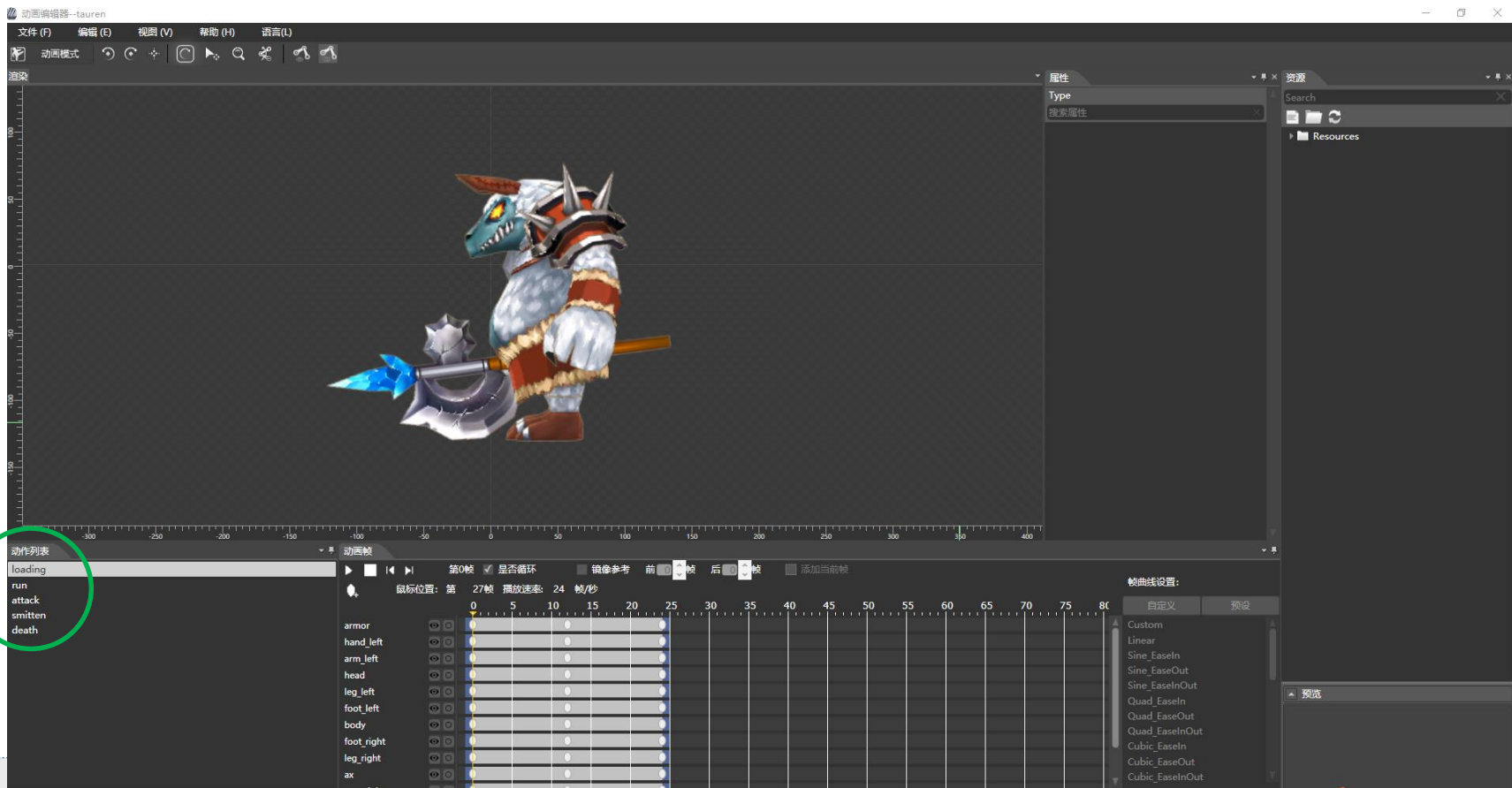
- Chapter 7

- 骨骼动画调用

- 游戏动画实例----侠客行

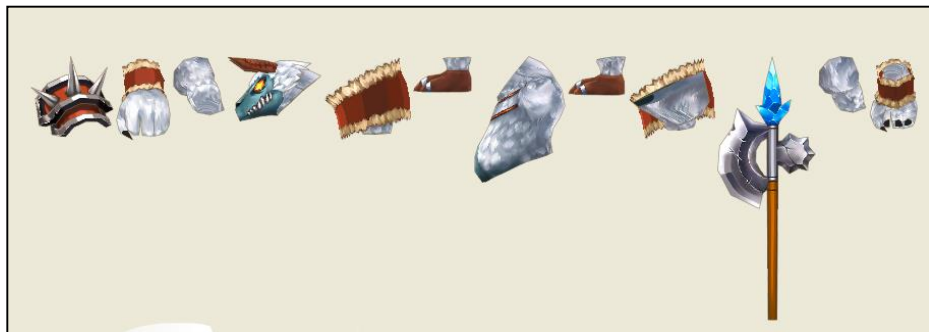
# 骨骼动画调用

- 本例使用的素材资源来源于cocos studio示例文件






# 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 > Resources > 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)

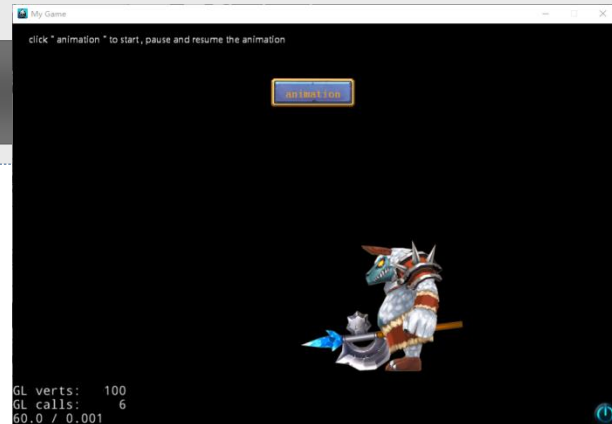
```
1 // 第六章例子1 -- 动画编辑器的应用
2
3 #ifndef __ANIMATION_EDITOR_SCENE_H__
4 #define __ANIMATION_EDITOR_SCENE_H__
5
6 // cocos2d
7 #include "cocos2d.h"
8 #include "cocostudio/CocoStudio.h"
9 #include "ui/CocosGUI.h"
10 using namespace cocos2d;
11 using namespace cocostudio;
12 using namespace cocos2d::ui;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



# 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->addTouchEventListeners(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;  
112     }  
113 }
```





**转场前**

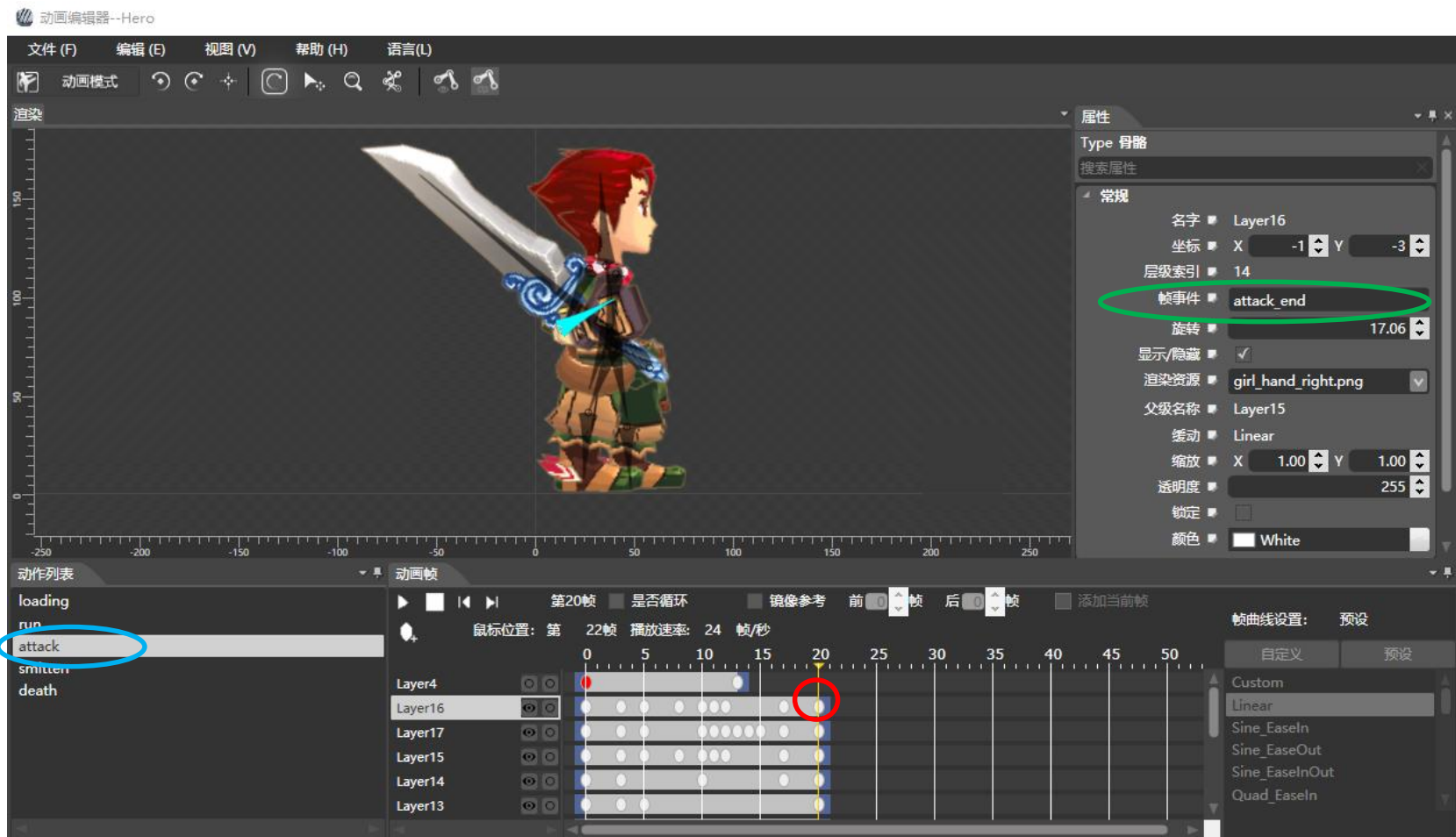
# 本节内容

- Chapter 7
  - 骨骼动画调用
  - 游戏动画实例----侠客行

# 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



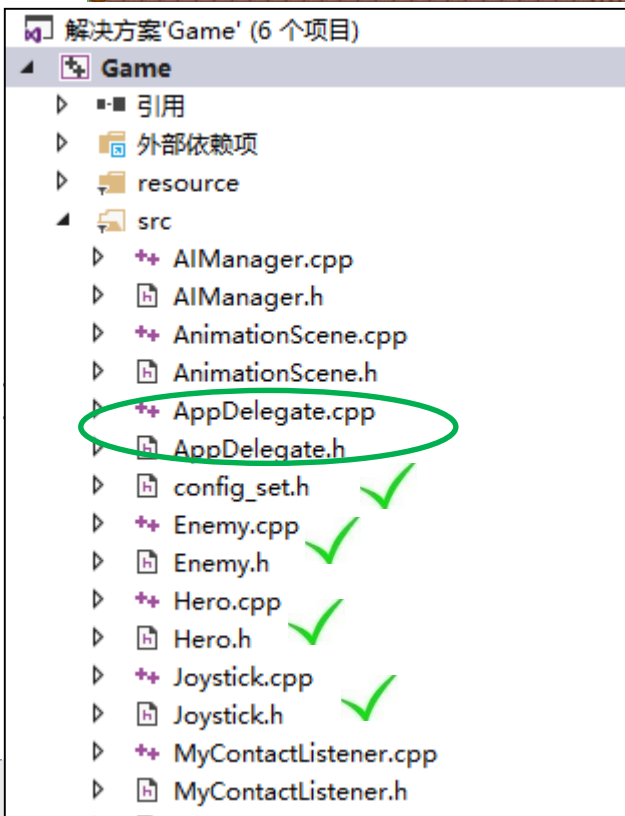
# 侠客行（教材demo）

**转场前**

# 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

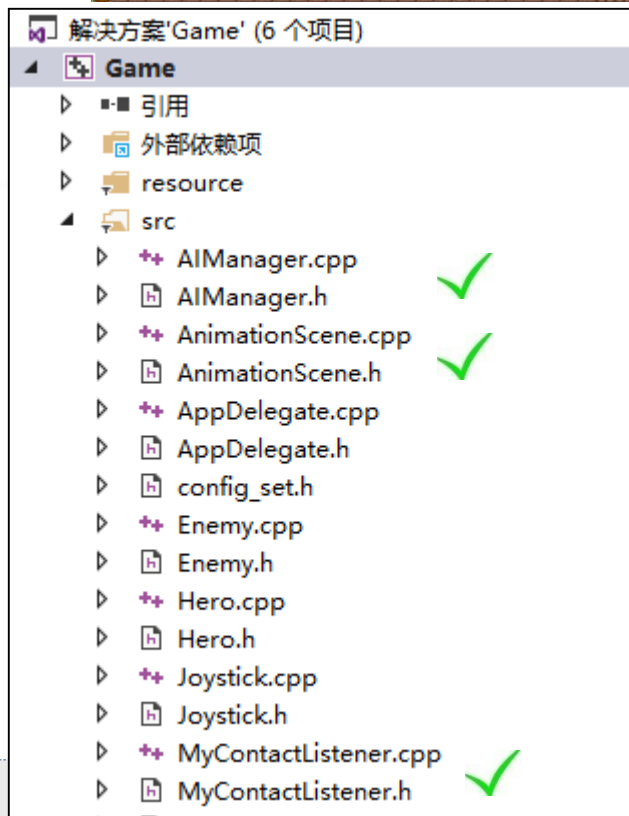
- AI管理类

AIManager

- 管理敌人的智能行为

- 主场景类

AnimationScene

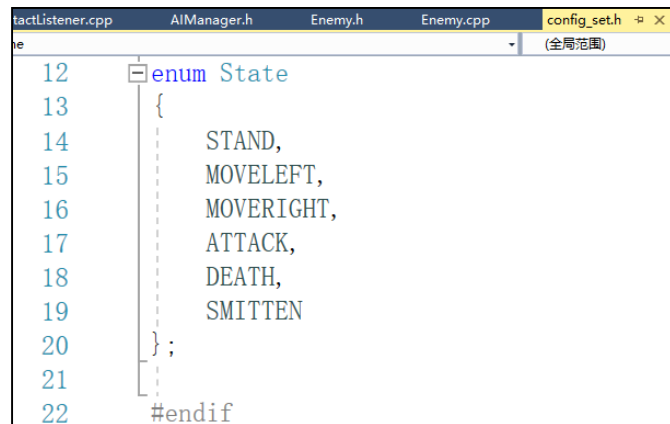




# 游戏动画实例——侠客行

## • 程序逻辑的实现

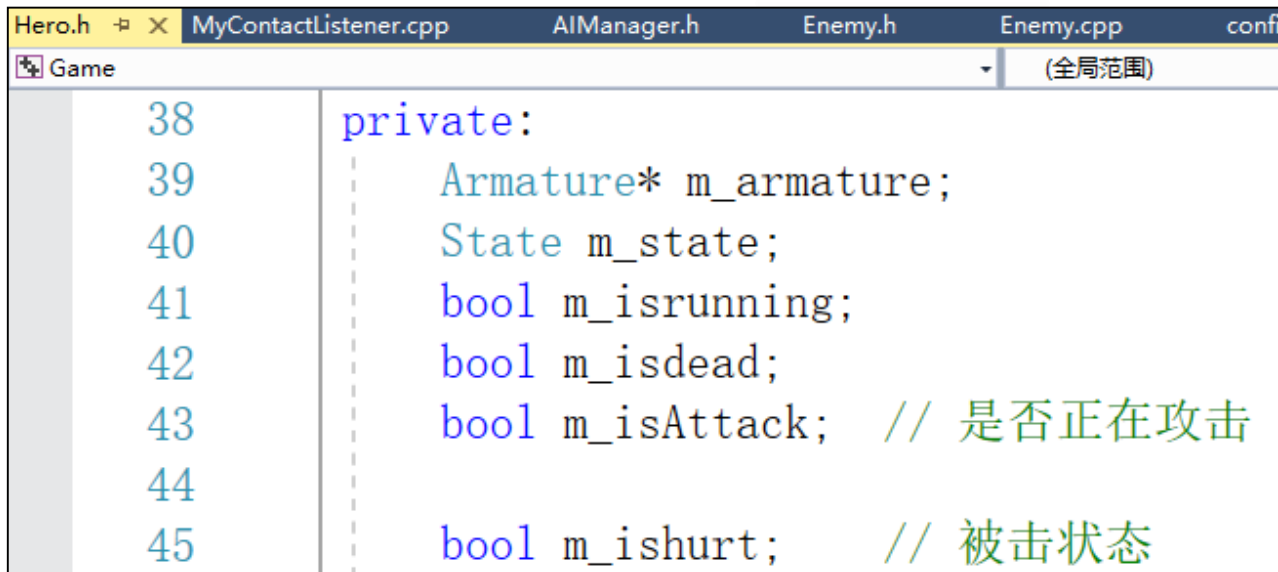
- 玩家和AI分别扮演英雄Hero和敌人Enemy
- 英雄和敌人的逻辑实际上是一样的，将会对玩家的操作反馈出站立、跑动（左右）、攻击、颤动、死亡等动画
- 这些动画播放的实现由一个状态量State来控制，包括 STAND、MOVE (LEFT、RIGHT)、ATTACK等



# 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



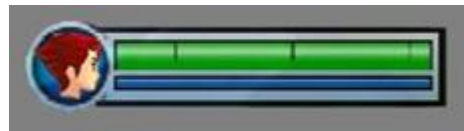
The screenshot shows a code editor with several tabs: Hero.h, MyContactListener.cpp (active), AIManager.h, Enemy.h, Enemy.cpp, and conf. The active file, MyContactListener.cpp, is in the 'Game' project and shows the private member variables of the Hero class. The variables are listed as follows:

```
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)
- 设置骨骼动画位置、播放待机动画
- 添加帧动画监听器
- 添加骨骼动画、设置英雄位置
- 启动调度器update



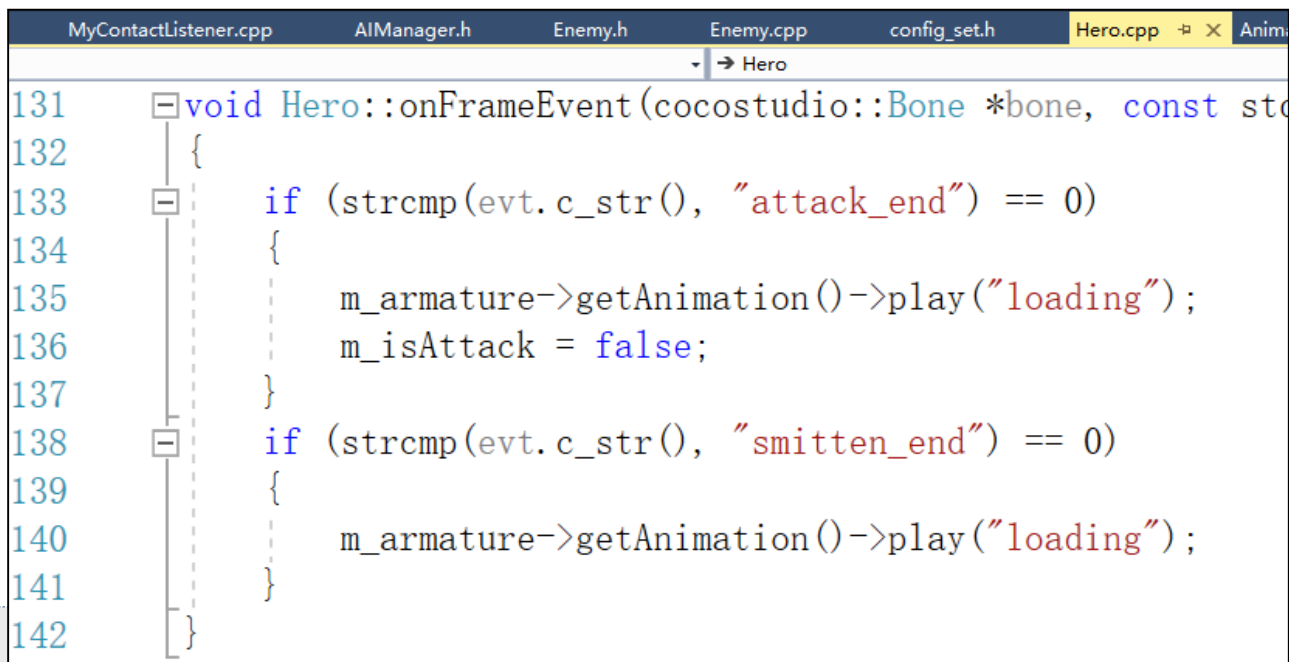
```
47 m_armature->setPosition(Vec2::ZERO);
48 m_armature->getAnimation()->play("loading");
49 m_armature->getAnimation()->setFrameEventCallFunc(CC_CALLBACK_0(Hero::onFrameEvent,
50 this->addChild(m_armature);
51 this->setPosition(position);
52
53 this->scheduleUpdate();
```



# 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
```

# 游戏动画实例——侠客行

## • MyContactListener类

- 继承自Node
- 成员变量
- 成员方法

```
class MyContactListener : public Node
```

```
Hero* m_hero;  
Enemy* m_enemy;
```

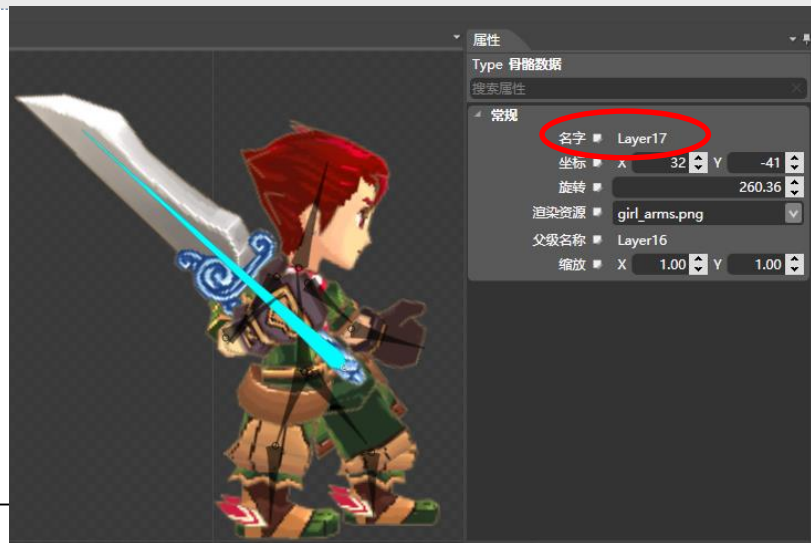


```
10 public:  
11     MyContactListener();  
12     ~MyContactListener();  
13  
14     static MyContactListener* create(Node* parent, Hero* hero, Enemy* enemy);  
15     virtual bool init(Node* parent, Hero* hero, Enemy* enemy);  
16     void update(float delta);  
17  
18     // set and get  
19     Hero* getHero() { return m_hero; }  
20     void setHero(Hero* hero) { m_hero = hero; }  
21  
22     Enemy* getEnemy() { return m_enemy; }  
23     void setEnemy(Enemy* enemy) { m_enemy = enemy; }
```



# 游戏动画实例——侠客行

- 重点是update()



```
55 void MyContactListener::update(float delta)
56 {
57     // hero 攻击 enemy
58     Vec2 hero_p_1 = m_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0, 0));
59     Vec2 hero_p_2 = m_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0, 15));
60     Vec2 hero_p_3 = m_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0, 30));
61     Vec2 hero_p_4 = m_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0, 50));
62     Rect enemy_rec = Rect(m_enemy->getPositionX(), m_enemy->getPositionY() - 40, 20, 40);
63     if (!m_enemy->isDeath() && m_hero->isAttack() && (enemy_rec.containsPoint(hero_p_1) || enemy_rec.containsPoint(hero_p_2)
64         || enemy_rec.containsPoint(hero_p_3) || enemy_rec.containsPoint(hero_p_4)))
65     {
66         // CCLOG("attack....enemy....");
67         m_enemy->hurt();
68         m_hero->setAttack(false);
69     }
```

# 游戏动画实例——侠客行

- **AIManager类**

- 继承自Node
- 成员变量

```
private:
    Enemy* m_enemy;
    //Hero* m_hero;

    State m_enemy_state;
```

- 成员方法

```
class AIManager : public Node
```

```
9      public:
10          AIManager();
11          ~AIManager();
12
13          static AIManager* create(Node* parent);
14          void setAI(Enemy* enemy, Hero* hero);
15
16          void moveLeft();
17          void moveRight();
18          void attack();
19          void stand();
20
21      private:
22          virtual bool init(Node* parent);
23          void update(float delta);
```

# 游戏动画实例——侠客行

- 重点是setAI(): 为m\_enemy创建一个永远播放的动作序列



```
34 void AIManager::setAI(Enemy* enemy, Hero* hero)
35 {
36     m_enemy = enemy;
37     //m_hero = hero;
38     this->scheduleUpdate();
39     auto sss = Sequence::create(
40         DelayTime::create(0.8f),
41         CallFunc::create(CC_CALLBACK_0(AIManager::moveLeft, this)),
42         DelayTime::create(1.0f),
43         CallFunc::create(CC_CALLBACK_0(AIManager::attack, this)),
44         DelayTime::create(0.3f),
45         CallFunc::create(CC_CALLBACK_0(AIManager::moveRight, this)),
46         DelayTime::create(0.7f),
47         CallFunc::create(CC_CALLBACK_0(AIManager::stand, this)),
48         DelayTime::create(0.5f),
49         CallFunc::create(CC_CALLBACK_0(AIManager::attack, this)),
50         NULL
51     );
52     auto act = RepeatForever::create(sss);
53     this->runAction(act);
54 }
```

# 游戏动画实例——侠客行

- 使用回调函数来完成具体的动作

```
132  void AIManager::moveLeft()  
133  {  
134      m_enemy_state = MOVELEFT;  
135  }
```

```
144  void AIManager::moveRight()  
145  {  
146      m_enemy_state = State::MOVERIGHT;  
147  }
```

```
156  void AIManager::attack()  
157  {  
158      m_enemy_state = State::ATTACK;  
159  }
```

```
168  void AIManager::stand()  
169  {  
170      m_enemy_state = State::STAND;  
171  }
```

# 游戏动画实例——侠客行

- `update()`: 根据`m_enemy_state`的值判断应该让敌人执行何种行为

```
83  void AIManager::update(float delta)
84  {
85      // 敌人动作
86      if (m_enemy_state == State::STAND)
87      {
88          m_enemy->play(STAND);
89      } else if (m_enemy_state == State::MOVELEFT)
90      {
91          m_enemy->play(MOVELEFT);
92      } else if (m_enemy_state == State::MOVERIGHT)
93      {
94          m_enemy->play(MOVERIGHT);
95      } else if (m_enemy_state == State::ATTACK)
96      {
97          m_enemy->play(ATTACK);
98      }
99  }
```

# 游戏动画实例——侠客行

- 场景类的实现:

- 生成背景

- 云朵
    - 背景文字
    - 英雄血条
    - 敌人血条
    - 英雄实例化
    - 敌人实例化
    - 摇杆
    - 攻击按钮



# 游戏动画实例——侠客行

- 场景类的实现：
  - 生成背景（略）
  - 攻击回调函数
    - 点击攻击按钮后调用



```
contactListener.h  Hero.h  MyContactListener.cpp  AIManager.h  Enemy.h  Enemy.cpp
ne  (全局范围)
160  void AnimationScene::attackCallback(Ref* pSender)
161  {
162      m_player->play(ATTACK);
163      m_enemy->play(ATTACK);
164  }
```

# 游戏动画实例——侠客行

- 场景类的实现:

- 生成背景
- 攻击回调函数
- update函数

- 更新云朵位置
- 更新英雄和敌人血条UI
- 将玩家的操作反馈到英雄的动画播放中



请自行阅读、学习、分析  
AnimationScene代码



# 游戏动画实例——侠客行

- 实验三任务1：找BUG
  - 英雄被攻击后不再接受玩家操作指令，卡在原地
  - 敌人大概率向左走，最后会撞左边的墙
  - 点击攻击按钮，英雄和敌人同时响应攻击操作
  - SMITTEN动作没有执行
  - ATTACK动作动画可以被打断
  - 有时候会出现没有打到但掉血的现象
  - 云朵移动动画出现衔接不上的现象
  - .....

# 游戏动画实例——侠客行

- 实验三任务2：优化功能
  - 修改1-2处英雄外型
  - 增加计分板
  - 优化敌人AI
  - 增加回合制功能



# 游戏动画实例——侠客行

## • 实验三提交文件要求:

— 实验报告

— 游戏录屏

— 源代码 (可选)

— 资源文件 (可选)

— 截止日期: 12.6周一晚23:59分

### 目录

一、实验目的与要求	2
二、实验内容与方法	2
三、实验步骤与过程	2
(一) 对本实验的分析	2
(二) config_set.h	3
(三) 英雄类 Hero.h 声明	3
1) Hero.h 成员变量	3
2) Hero.h 成员方法	4
(四) 英雄类 Hero.cpp 实现	4
1) 构造函数 Hero::Hero()	4
2) Hero 精灵初始化函数 Hero::init(Vec2 position)	5
3) Hero::update(float dt)部分	6
4) Hero::play()函数	7
5) Hero::hurt()及其相关函数	8
6) Hero 和 Enemy 类改进	9
(五) 碰撞检测组件 ContactListener	10
1) MyContactListener.h 类声明	10
2) 构造函数、create 函数、初始化 init 函数	11
3) update()函数	12
(六) 摇杆类 Joystick 封装	13
(七) 敌人 AI 管理器	13
1) AIManager.h 头文件说明	14
2) AIManager 的具体实现	14
(八) 场景类 AnimationScene 的实现	15
1) 背景的生成	15
2) 攻击回调函数	17
3) 场景类的 update 函数	17
(九) 改进方案	20
1) 方案 1: 修改英雄动画	20
2) 方案 2: 增加比率和重开功能	22
3) 方案 3 (拟): 增加防御动作和相关逻辑处理	24
四、实验结论或心得体会	27