1. **二维游戏动画合成**

**侠客行**

## **（一）动画编辑器的使用方法**

Cocos2d-x 自带一个功能强大的免费工具集CocosStudio，它包含了游戏开发中所需要的四大功能工具：动画编辑器、UI编辑器、数据编辑器和场景编辑器，本书中使用的是v1.3.0.1版本。

动画编辑器：用于编辑游戏中使用的角色动画、特效动画、场景动画等动态的游戏资源。动画编辑器支持碎图导出，整图导出等多种资源优化方式，同时也支持关键帧动画，序列帧动画，骨骼动画等多种动画编辑方式。动画编辑器支持对Flash，PSD，Plist资源的解析。下面我们通过制作一个简单的动画来带大家初步认识CocosStudio。

1. 创建项目

下载安装好CocosStudio后，打开CocosStudio将出现如下界面：



图6-3 CocosStudio运行界面

选择Animation Editor->文件->新建，创建新项目，这里我们命名为UI\_Example，创建好项目之后将会出现图6-4所示界面，界面左上角显示的是画布大小。

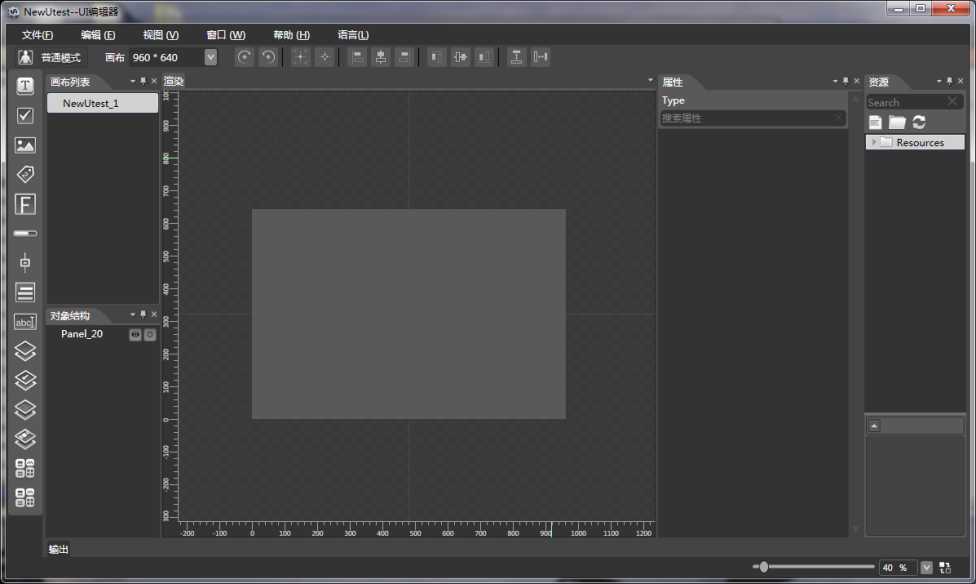


图6-4 Animation Editor界面

1. 导入资源

在新创建项目资源面板点击鼠标右键即可导入资源，导入后资源面板状态如图6-6所示：

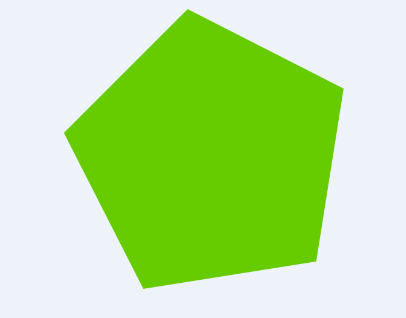
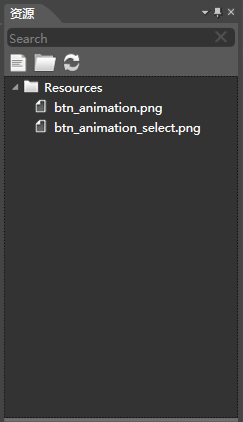
 

图6-5 资源图片button.png 图6-6资源面板

1. 编辑动画

1）渲染窗口有两种模式，形体模式和动画模式。

形体模式：用户可以在该模式下组装角色部件或绑定骨骼。

动画模式：编辑、调整和生成动画。

2）在形体模式将导入的五边形图片素材拖放到渲染窗口，复制2份，将三个五边形分别命名为p1,p2,p3摆出如图形状：

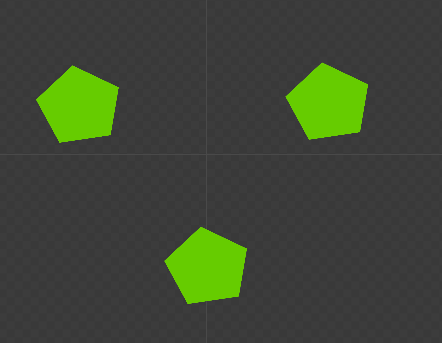
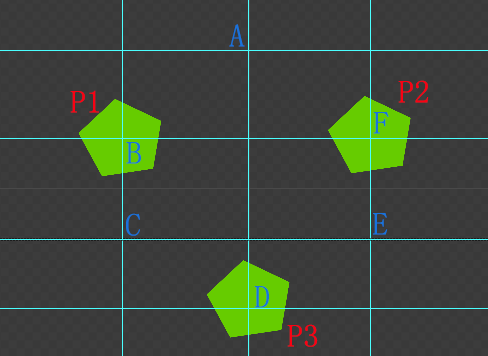


图6-7 p1,p2,p3

3）切换进入动画模式

在渲染区拉出参考线如图，交点即为A,B,C,D,E,F:



在动画帧区，在时间轴p1,p2,p3上分别都在15帧和30帧处用鼠标右键添加关键帧：

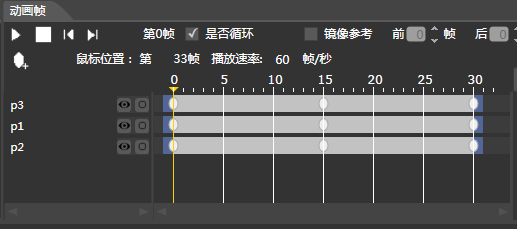


图6-9 添加动画帧

在时间轴上做如下操作：

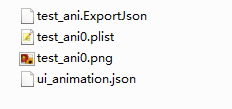
a.鼠标左键点击选中第15帧关键帧，此时把p1移动到C点，p2移动到A点，p3移动到E点；

b.然后再选中时间轴第30帧关键帧，此时把p1移动到D点，p2移动到B点，p3移动到F点；

到此动画设定完成，可以点击动画帧区播放键预览动画。

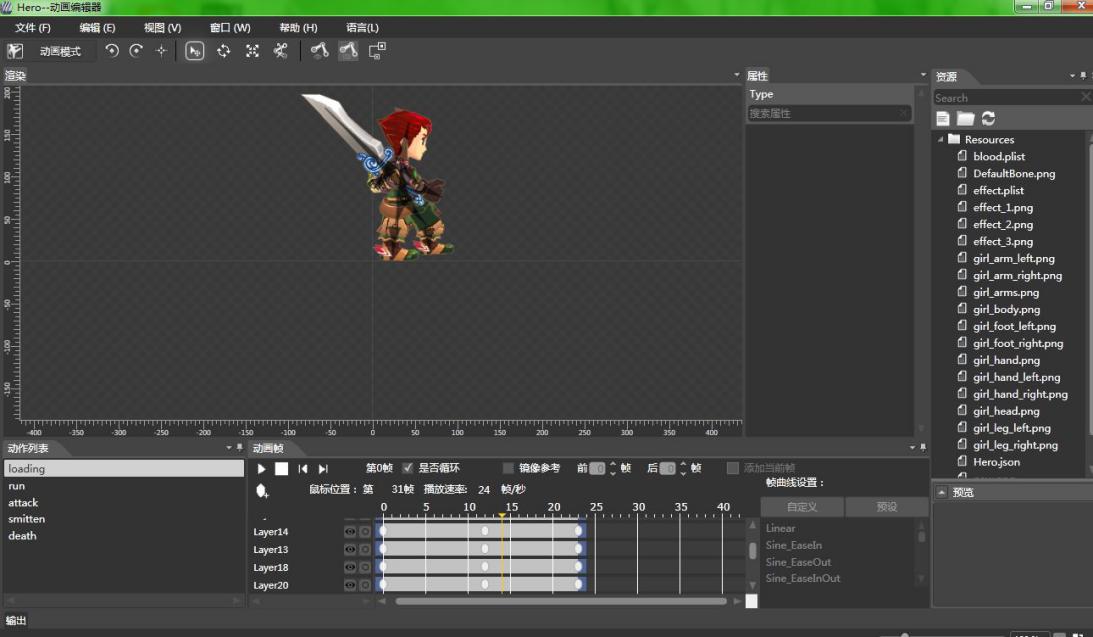
1. 导出动画

选择文件->导出项目 使用默认参数即可导出动画，打开发布目录，文件结构如下图，需要使用时需要将所有文件放在同一路径。



**（二）本实例的骨骼动画**

本例使用的素材资源来源于cocos2d-x官方提供的cpptest中的资源，打开引擎中cpptest下的Resources目录可以找到



人物现有的动作可以不经过修改就能满足项目的需求，但是需要在一些动作的末尾添加帧事件:

* + - 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

以同样的方式对怪物也做相同的处理。

**（三）程序的实现**

头文件包含及常量的定义放在单独的文件config\_set.h内：

#ifndef \_\_CONFIG\_SET\_H\_\_

#define \_\_CONFIG\_SET\_H\_\_

#include "cocos2d.h"

#include "cocostudio/CocoStudio.h"

#include "ui/CocosGUI.h"

USING\_NS\_CC;

using namespace cocos2d;

using namespace cocostudio;

using namespace cocos2d::ui;

enum State

{

STAND,

MOVELEFT,

MOVERIGHT,

ATTACK,

DEATH,

SMITTEN

};

#endif

封装英雄类：Hero

英雄类的内部实现需要注意的地方是处理受到攻击后播放受攻击动画的末尾，应该触发帧事件，然后让英雄回归到正常的待机状态；同样，在英雄跑动结束时，也应该回归正常的待机状态，具体实现：

#ifndef \_\_HERO\_H\_\_ // Hero.h

#define \_\_HERO\_H\_\_

#include "config\_set.h"

class Hero : public Sprite

{

public:

Hero();

~Hero();

static Hero\* create(Vec2 position);

void update(float delta);

void play(State state);

void hurt(); // 被击中

void showBloodTips(int s); // 显示扣血数字

void flyend(Label\* label); // 扣血数字移动并消失

virtual bool init(Vec2 position);

void onFrameEvent(cocostudio::Bone \*bone, const std::*string*& evt, int originFrameIndex, int currentFrameIndex);

// set and get

Armature\* getArmature() { return m\_armature; }

bool isAttack() { return m\_isAttack; }

void setAttack(bool attack) { m\_isAttack = attack; }

int getLife() { return m\_life; }

void setLife(int life) { m\_life = life; }

bool isDeath() { return m\_isdead; }

int getMaxLife() { return m\_max\_life; }

private:

Armature\* m\_armature;

State m\_state;

bool m\_isrunning;

bool m\_isdead;

bool m\_isAttack; // 是否正在攻击

bool m\_ishurt; // 被击状态

int m\_life; // 生命值

int m\_max\_life; // 最大生命值

};

#endif

实现文件为Hero.cpp

#include "Hero.h"

Hero::Hero()

{

m\_isrunning = false;

m\_isdead = false;

m\_isAttack = false;

m\_ishurt = false;

m\_max\_life = 500;

m\_life = m\_max\_life;

}

Hero::~Hero()

{

}

Hero\* Hero::create(Vec2 position)

{

Hero \*pRet = new(std::*nothrow*) Hero();

if (pRet && pRet->init(position))

{

pRet->autorelease();

return pRet;

} else

{

delete pRet;

pRet = NULL;

return NULL;

}

}

bool Hero::init(Vec2 position)

{

if (!Sprite::init())

{

return false;

}

ArmatureDataManager::getInstance()->addArmatureFileInfo("Chapter06/AnimationScene/animation/Hero/Hero.ExportJson");

m\_armature = Armature::create("Hero");

if (m\_armature == NULL)

{

CCLOG("hero load error!");

return false;

}

m\_armature->setPosition(Vec2::ZERO);

m\_armature->getAnimation()->play("loading");

m\_armature->getAnimation()->setFrameEventCallFunc(CC\_CALLBACK\_0(Hero::onFrameEvent, this, std::*placeholders*::*\_1*, std::*placeholders*::*\_2*, std::*placeholders*::*\_3*, std::*placeholders*::*\_4*));

this->addChild(m\_armature);

this->setPosition(position);

this->scheduleUpdate();

return true;

}

void Hero::update(float delta)

{

if (m\_life <= 0)

{

play(DEATH);

}

switch (m\_state)

{

case STAND:

if ((m\_isrunning == true) && (m\_isdead == false)&& (m\_ishurt == false)) // 跑动结束只执行一次loading动画，避免点击attack动作无法播放

{

m\_armature->getAnimation()->play("loading");

m\_isrunning = false;

}

break;

case MOVELEFT:

if ((this->getPositionX() > 0) && (m\_isdead == false)&& (m\_ishurt == false) && (m\_isAttack == false))

{

if (m\_isrunning == false)

{

m\_armature->getAnimation()->play("run");

m\_isrunning = true;

}

if (m\_armature->getScaleX() != -1)

{

m\_armature->setScaleX(-1);

}

this->setPositionX(this->getPositionX() - 4);

}

break;

case MOVERIGHT:

if ((m\_isAttack == false) && (this->getPositionX() < Director::getInstance()->getVisibleSize().width) && (m\_isdead == false)&& (m\_ishurt == false))

{

if (m\_isrunning == false) // m\_isrunning控制变量防止update时不断执行play("run")则永远显示run动画第一帧

{

m\_armature->getAnimation()->play("run");

m\_isrunning = true;

}

if (m\_armature->getScaleX() != 1.0)

{

m\_armature->setScaleX(1);

}

this->setPositionX(this->getPositionX() + 4);

}

break;

case ATTACK:

if (m\_isdead == false&& (m\_ishurt == false))

{

m\_isAttack = true;

m\_armature->getAnimation()->play("attack");

}

break;

case DEATH:

if (m\_isdead == false)

{

m\_isdead = true;

m\_armature->getAnimation()->play("death");

}

break;

case SMITTEN:

if (m\_isdead == false)

{

if (m\_ishurt == true)

{

m\_armature->getAnimation()->play("smitten");

m\_ishurt = false;

}

}

break;

}

}

void Hero::onFrameEvent(cocostudio::Bone \*bone, const std::*string*& evt, int originFrameIndex, int currentFrameIndex)

{

if (*strcmp*(evt.*c\_str*(), "attack\_end") == 0)

{

m\_armature->getAnimation()->play("loading");

m\_isAttack = false;

}

if (*strcmp*(evt.*c\_str*(), "smitten\_end") == 0)

{

m\_armature->getAnimation()->play("loading");

}

}

void Hero::play(State state)

{

if (state == SMITTEN) // 控制被击中时颤抖动画只播放一次

{

m\_ishurt = true;

}

m\_state = state;

}

void Hero::hurt()

{

// 根据基础伤害造成随机伤害

showBloodTips(30);

this->play(SMITTEN);

}

// 显示扣血

void Hero::showBloodTips(int s)

{

int hitCount = 1;

int hitRand = *rand*()%10;

if (hitRand > 3 && hitRand < 8)

{

hitCount = 2;

} else if (hitRand > 7)

{

hitCount = 3;

}

for (int i = 0; i < hitCount; i ++)

{

int hurt\_blood = s + *rand*()%8;

setLife(m\_life - hurt\_blood); // 扣血

auto label = Label::createWithBMFont("fonts/futura-48.fnt", StringUtils::format("-%d",hurt\_blood));

label->setColor(Color3B::RED);

this->addChild(label, 5);

label->setPosition(Vec2(0, 0) + Vec2(20 + *rand*()%80, 10 + *rand*()%80));

label->runAction(Sequence::create(

MoveBy::create(0.7f,Vec2(0,30)),

CallFunc::create(CC\_CALLBACK\_0(Hero::flyend, this, label)),

NULL

));

}

}

void Hero::flyend(Label\* label)

{

label->setVisible(false);

label->removeFromParent();

}

接下来怪物类的封装和英雄类类似：

#ifndef \_\_ENEMY\_H\_\_ // Ememy.h

#define \_\_ENEMY\_H\_\_

#include "config\_set.h"

class Enemy : public Sprite

{

public:

Enemy();

~Enemy();

static Enemy\* create(Vec2 position);

void update(float delta);

void play(State state); // 播放动画

void hurt(); // 被击中

void showBloodTips(int s); // 显示扣血数字

void flyend(Label\* label); // 扣血数字移动并消失

virtual bool init(Vec2 position);

void onFrameEvent(cocostudio::Bone \*bone, const std::*string*& evt, int originFrameIndex, int currentFrameIndex);

// set and get

Armature\* getArmature() { return m\_armature; }

bool isAttack() { return m\_isAttack; }

void setAttack(bool attack) { m\_isAttack = attack; }

int getLife() { return m\_life; }

void setLife(int life) { m\_life = life; }

bool isDeath() { return m\_isdead; }

int getMaxLife() { return m\_max\_life; }

private:

Armature\* m\_armature;

State m\_state; // 人物动画状态

bool m\_isrunning; // 是否正在跑动

bool m\_isdead; // 是否死亡

bool m\_isAttack; // 攻击状态

bool m\_ishurt; // 被击状态

int m\_life; // 生命值

int m\_max\_life; // 最大生命值

};

#endif

实现文件为：

#include "Enemy.h" // Emeny.cpp

Enemy::Enemy()

{

m\_isrunning = false;

m\_isdead = false;

m\_isAttack = false;

m\_ishurt = false;

m\_max\_life = 500;

m\_life = m\_max\_life;

}

Enemy::~Enemy()

{

}

Enemy\* Enemy::create(Vec2 position)

{

Enemy \*pRet = new(std::*nothrow*) Enemy();

if (pRet && pRet->init(position))

{

pRet->autorelease();

return pRet;

} else

{

delete pRet;

pRet = NULL;

return NULL;

}

}

bool Enemy::init(Vec2 position)

{

if (!Sprite::init())

{

return false;

}

ArmatureDataManager::getInstance()->addArmatureFileInfo("Chapter06/AnimationScene/animation/tauren/tauren.ExportJson");

m\_armature = Armature::create("tauren");

if (m\_armature == NULL)

{

CCLOG("tauren load error!");

return false;

}

m\_armature->setPosition(Vec2::ZERO);

m\_armature->getAnimation()->play("loading");

m\_armature->getAnimation()->setFrameEventCallFunc(CC\_CALLBACK\_0(Enemy::onFrameEvent, this, std::*placeholders*::*\_1*, std::*placeholders*::*\_2*, std::*placeholders*::*\_3*, std::*placeholders*::*\_4*));

this->addChild(m\_armature);

this->setPosition(position);

this->scheduleUpdate();

return true;

}

void Enemy::update(float delta)

{

if (m\_life <= 0)

{

play(DEATH);

}

switch (m\_state)

{

case STAND:

if ((m\_isAttack == false) && (m\_isrunning == true) && (m\_isdead == false) && (m\_ishurt == false)) // 跑动结束只执行一次loading动画，避免点击attack动作无法播放

{

m\_armature->getAnimation()->play("loading");

m\_isrunning = false;

}

break;

case MOVELEFT:

if ((m\_isAttack == false) && (this->getPositionX() > 0) && (m\_isdead == false) && (m\_ishurt == false))

{

if (m\_isrunning == false)

{

m\_armature->getAnimation()->play("run");

m\_isrunning = true;

}

if (m\_armature->getScaleX() != 1)

{

m\_armature->setScaleX(1);

}

this->setPositionX(this->getPositionX() - 3);

}

break;

case MOVERIGHT:

if ((m\_isAttack == false) && (this->getPositionX() < Director::getInstance()->getVisibleSize().width) && (m\_isdead == false) && (m\_ishurt == false))

{

if (m\_isrunning == false) // m\_isrunning控制变量防止update时不断执行play("run")则永远显示run动画第一帧

{

m\_armature->getAnimation()->play("run");

m\_isrunning = true;

}

if (m\_armature->getScaleX() != -1.0)

{

m\_armature->setScaleX(-1);

}

this->setPositionX(this->getPositionX() + 3);

}

break;

case ATTACK:

if (m\_isdead == false && (m\_ishurt == false))

{

m\_isAttack = true;

m\_armature->getAnimation()->play("attack");

}

break;

case DEATH:

if (m\_isdead == false)

{

m\_isdead = true;

m\_armature->getAnimation()->play("death");

}

break;

case SMITTEN:

if (m\_isdead == false)

{

if (m\_ishurt == true)

{

m\_armature->getAnimation()->play("smitten");

m\_ishurt = false;

}

}

break;

}

}

void Enemy::onFrameEvent(cocostudio::Bone \*bone, const std::*string*& evt, int originFrameIndex, int currentFrameIndex)

{

if (*strcmp*(evt.*c\_str*(), "attack\_end") == 0)

{

m\_armature->getAnimation()->play("loading");

m\_isAttack = false;

}

if (*strcmp*(evt.*c\_str*(), "smitten\_end") == 0)

{

m\_armature->getAnimation()->play("loading");

}

}

void Enemy::play(State state)

{

if (state == SMITTEN)

{

m\_ishurt = true;

}

m\_state = state;

}

void Enemy::hurt()

{

// 根据基础伤害造成随机伤害

showBloodTips(30);

play(SMITTEN);

}

// 显示扣血

void Enemy::showBloodTips(int s)

{

int hitCount = 1;

int hitRand = *rand*()%10;

if (hitRand > 3 && hitRand < 8)

{

hitCount = 2;

} else if (hitRand > 7)

{

hitCount = 3;

}

for (int i = 0; i < hitCount; i ++)

{

int hurt\_blood = s + *rand*()%8;

setLife(m\_life - hurt\_blood); // 扣血

auto label = Label::createWithBMFont("fonts/futura-48.fnt", StringUtils::format("-%d",hurt\_blood));

this->addChild(label, 5);

label->setPosition(Vec2(0, 0) + Vec2(20 + *rand*()%80, 10 + *rand*()%80));

label->runAction(Sequence::create(

MoveBy::create(0.7f,Vec2(0,30)),

CallFunc::create(CC\_CALLBACK\_0(Enemy::flyend, this, label)),

NULL

));

}

}

void Enemy::flyend(Label\* label)

{

label->setVisible(false);

label->removeFromParent();

}

摇杆类的封装Joystick:

头文件：

// 摇杆 // Joystick.h

#ifndef \_\_JOYSTICK\_H\_\_

#define \_\_JOYSTICK\_H\_\_

#include "config\_set.h"

enum Joystick\_dir

{

\_LEFT,

\_RIGHT,

\_STOP

};

class Joystick : public Layer

{

public:

Joystick();

~Joystick();

// 初始化 aPoint是摇杆中心 aRadius是摇杆半径 aJsSprite是摇杆控制点 aJsBg是摇杆背景

static Joystick\* create(Vec2 aPoint ,float aRadius ,char\* aJsSprite,char\* aJsBg);

// 获取摇杆方向

//Vec2 getDirection();

Joystick\_dir getDirection();

private:

float getVelocity(); // 摇杆力度

void update(float dt);

virtual bool init(Vec2 aPoint ,float aRadius ,char\* aJsSprite,char\* aJsBg);

virtual bool onTouchBegan(Touch \*pTouch, Event \*pEvent);

virtual void onTouchMoved(Touch \*pTouch, Event \*pEvent);

virtual void onTouchEnded(Touch \*pTouch, Event \*pEvent);

private:

Vec2 m\_centerPoint; // 摇杆中心

Vec2 m\_currentPoint; // 摇杆当前位置

float m\_radius; // 摇杆半径

Sprite\* m\_jsSprite;

};

#endif

实现文件:

#include "Joystick.h"

Joystick::Joystick()

{}

Joystick::~Joystick()

{}

bool Joystick::init(Vec2 aPoint , float aRadius , char\* aJsSprite, char\* aJsBg)

{

if (!Layer::init())

{

return false;

}

m\_radius = aRadius;

m\_centerPoint = aPoint;

m\_currentPoint = m\_centerPoint;

m\_jsSprite = Sprite::create(aJsSprite);

m\_jsSprite->setPosition(m\_centerPoint);

auto \_aJsBg = Sprite::create(aJsBg);

\_aJsBg->setPosition(m\_centerPoint);

\_aJsBg->setTag(88);

this->addChild(\_aJsBg);

this->addChild(m\_jsSprite);

this->scheduleUpdate();

auto listener = EventListenerTouchOneByOne::create();

listener->onTouchBegan = CC\_CALLBACK\_2(Joystick::onTouchBegan, this);

listener->onTouchMoved = CC\_CALLBACK\_2(Joystick::onTouchMoved, this);

listener->onTouchEnded = CC\_CALLBACK\_2(Joystick::onTouchEnded, this);

Director::getInstance()->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener, this);

return true;

}

Joystick\* Joystick::create(Vec2 aPoint, float aRadius, char\* aJsSprite, char\* aJsBg)

{

Joystick \*pRet = new(std::*nothrow*) Joystick();

if (pRet && pRet->init(aPoint, aRadius, aJsSprite, aJsBg)) {

pRet->autorelease();

return pRet;

}

else {

delete pRet;

pRet = NULL;

return NULL;

}

}

// 获取摇杆当前方向

Joystick\_dir Joystick::getDirection()

{

if ((m\_currentPoint - m\_centerPoint).x > 0)

{

return Joystick\_dir::\_RIGHT;

} else if ((m\_currentPoint - m\_centerPoint).x < 0)

{

return Joystick\_dir::\_LEFT;

}

return Joystick\_dir::\_STOP;

}

// 获取摇杆力度

float Joystick::getVelocity()

{

return m\_centerPoint.getDistance(m\_currentPoint);

}

// 更新摇杆按钮位置

void Joystick::update(float dt)

{

m\_jsSprite->setPosition(m\_jsSprite->getPosition() + (m\_currentPoint - m\_jsSprite->getPosition()) \* 0.5);

}

bool Joystick::onTouchBegan(Touch \*pTouch, Event \*pEvent)

{

auto touchPoint = pTouch->getLocation();

if (touchPoint.getDistance(m\_centerPoint) > m\_radius){

return false;

}

m\_currentPoint = touchPoint;

return true;

}

void Joystick::onTouchMoved(Touch \*pTouch, Event \*pEvent)

{

auto touchPoint = pTouch->getLocation();

if (touchPoint.getDistance(m\_centerPoint) > m\_radius)

{

m\_currentPoint = m\_centerPoint + (touchPoint - m\_centerPoint).getNormalized() \* m\_radius;

}else {

m\_currentPoint = touchPoint;

}

}

void Joystick::onTouchEnded(Touch \*pTouch, Event \*pEvent)

{

m\_currentPoint = m\_centerPoint;

}

还需要一个碰撞检测的类来同一管理游戏场景的碰撞检测：

#ifndef \_\_CONTACTLISTENER\_H\_\_ // MyContactListener.h

#define \_\_CONTACTLISTENER\_H\_\_

#include "config\_set.h"

#include "Hero.h"

#include "Enemy.h"

class MyContactListener : public Node

{

public:

MyContactListener();

~MyContactListener();

static MyContactListener\* create(Node\* parent, Hero\* hero, Enemy\* enemy);

virtual bool init(Node\* parent, Hero\* hero, Enemy\* enemy);

void update(float delta);

// set and get

Hero\* getHero() { return m\_hero; }

void setHero(Hero\* hero) { m\_hero = hero; }

Enemy\* getEnemy() { return m\_enemy; }

void setEnemy(Enemy\* enemy) { m\_enemy = enemy; }

private:

Vector<Sprite\*> m\_contactList;

Hero\* m\_hero;

Enemy\* m\_enemy;

};

#endif

碰撞方法中主要检测场景中的Hero和Ememy各自的武器是否已经触碰到对方，如果触碰到对方，则触发一次扣血操作，有一定几率触发暴击，具体实现如下：

#include "MyContactListener.h"

MyContactListener::MyContactListener()

{

}

MyContactListener::~MyContactListener()

{

}

bool MyContactListener::init(Node\* parent, Hero\* hero, Enemy\* enemy)

{

if (!Node::init())

{

return false;

}

m\_hero = hero;

m\_enemy = enemy;

parent->addChild(this);

this->scheduleUpdate();

return true;

}

void MyContactListener::update(float delta)

{

// hero 攻击 enemy

Vec2 hero\_p\_1 = m\_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0,0));

Vec2 hero\_p\_2 = m\_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0,15));

Vec2 hero\_p\_3 = m\_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0,30));

Vec2 hero\_p\_4 = m\_hero->getArmature()->getBone("Layer17")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0,50));

Rect enemy\_rec = Rect(m\_enemy->getPositionX(),m\_enemy->getPositionY() - 40, 20, 40);

if (!m\_enemy->isDeath() && m\_hero->isAttack() && (enemy\_rec.containsPoint(hero\_p\_1) || enemy\_rec.containsPoint(hero\_p\_2)

|| enemy\_rec.containsPoint(hero\_p\_3) || enemy\_rec.containsPoint(hero\_p\_4)))

{

// CCLOG("attack....enemy....");

m\_enemy->hurt();

m\_hero->setAttack(false);

}

// enemy 攻击 hero

Vec2 enemy\_p\_1 = m\_enemy->getArmature()->getBone("ax")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0,0));

Vec2 enemy\_p\_2 = m\_enemy->getArmature()->getBone("ax")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0,30));

Rect hero\_rec = Rect(m\_hero->getPositionX() - 10,m\_hero->getPositionY() - 20,20,40);

if (!m\_hero->isDeath() && m\_enemy->isAttack() &&(hero\_rec.containsPoint(enemy\_p\_1) || hero\_rec.containsPoint(enemy\_p\_2)))

{

// CCLOG("attack...hero....");

m\_hero->hurt();

m\_enemy->setAttack(false);

}

}

MyContactListener\* MyContactListener::create(Node\* parent, Hero\* hero, Enemy\* enemy)

{

MyContactListener \*pRet = new(std::*nothrow*) MyContactListener();

if (pRet && pRet->init(parent, hero, enemy))

{

pRet->autorelease();

return pRet;

} else

{

delete pRet;

pRet = NULL;

return NULL;

}

}

在创建主场景之前，我们还需要一个AIManager来管理场景中的Ememy的职能行为，让Ememy有一定的机动性，不过这里给出的只是一个简单的动作序列，如果想要给怪物加上更加职能的行为，需要自己实现怪物的行为算法，这里不进行过多的扩展。实现如下：

#ifndef \_\_AIMANAGER\_H\_\_ // AIManager.h

#define \_\_AIMANAGER\_H\_\_

#include "config\_set.h"

#include "Enemy.h"

#include "Hero.h"

class AIManager : public Node

{

public:

AIManager();

~AIManager();

static AIManager\* create(Node\* parent);

void setAI(Enemy\* enemy, Hero\* hero);

void moveLeft();

void moveRight();

void attack();

void stand();

private:

virtual bool init(Node\* parent);

void update(float delta);

private:

Enemy\* m\_enemy;

Hero\* m\_hero;

State m\_enemy\_state;

};

#endif

实现文件如下：

#include "AIManager.h"

AIManager\* AIManager::create(Node\* parent)

{

AIManager\* pRet = new(std::*nothrow*) AIManager();

if (pRet && pRet->init(parent))

{

pRet->autorelease();

return pRet;

} else

{

delete pRet;

pRet = NULL;

return NULL;

}

}

void AIManager::setAI(Enemy\* enemy, Hero\* hero)

{

m\_enemy = enemy;

m\_hero = hero;

this->scheduleUpdate();

auto sss = Sequence::create(

DelayTime::create(0.8f),

CallFunc::create(CC\_CALLBACK\_0(AIManager::moveLeft, this)),

DelayTime::create(1.0f),

CallFunc::create(CC\_CALLBACK\_0(AIManager::attack, this)),

DelayTime::create(0.3f),

CallFunc::create(CC\_CALLBACK\_0(AIManager::moveRight, this)),

DelayTime::create(0.7f),

CallFunc::create(CC\_CALLBACK\_0(AIManager::stand, this)),

DelayTime::create(0.5f),

CallFunc::create(CC\_CALLBACK\_0(AIManager::attack, this)),

NULL

);

auto act = RepeatForever::create(sss);

this->runAction(act);

}

bool AIManager::init(Node\* parent)

{

if (!Node::init())

{

return false;

}

parent->addChild(this);

return true;

}

void AIManager::update(float delta)

{

// 敌人动作

if (m\_enemy\_state == State::STAND)

{

m\_enemy->play(STAND);

} else if (m\_enemy\_state == State::MOVELEFT)

{

m\_enemy->play(MOVELEFT);

} else if (m\_enemy\_state == State::MOVERIGHT)

{

m\_enemy->play(MOVERIGHT);

} else if (m\_enemy\_state == State::ATTACK)

{

m\_enemy->play(ATTACK);

}

}

AIManager::AIManager()

{

m\_enemy\_state = State::MOVELEFT;

}

AIManager::~AIManager()

{}

void AIManager::moveLeft()

{

m\_enemy\_state = MOVELEFT;

}

void AIManager::moveRight()

{

m\_enemy\_state = State::MOVERIGHT;

}

void AIManager::attack()

{

m\_enemy\_state = State::ATTACK;

}

void AIManager::stand()

{

m\_enemy\_state = State::STAND;

}

现在可以搭建主场景了，加入主场景类：

// 第6章例子2 -- 综合动画例子

#ifndef \_\_ANIMATION\_SCENE\_H\_\_ // AnimationScene.h

#define \_\_ANIMATION\_SCENE\_H\_\_

#include "config\_set.h"

#include "Joystick.h"

#include "Hero.h"

#include "Enemy.h"

#include "MyContactListener.h"

#include "AIManager.h"

class AnimationScene : public Layer

{

public:

AnimationScene();

~AnimationScene();

virtual bool init();

CREATE\_FUNC(AnimationScene);

static Scene\* createScene();

void update(float delta);

void attackCallback(Ref\* pSender);

private:

Joystick\* m\_joystick; // 摇杆

MyContactListener\* m\_contactListener; // 碰撞检测

AIManager\* m\_aimanager; // AI manager

Hero\* m\_player;

Enemy\* m\_enemy;

Sprite\* m\_bloodBar\_h;

Sprite\* m\_bloodBar\_e;

};

#endif

在实现文件中创建场景，加入任务和怪物，并进行控制英雄处理、添加怪物AI等操作：

#include "AnimationScene.h" // AnimationScene.cpp

AnimationScene::AnimationScene()

{

}

AnimationScene::~AnimationScene()

{

}

bool AnimationScene::init()

{

if (!Layer::init())

{

return false;

}

Size visibleSize = Director::getInstance()->getVisibleSize();

// 背景

auto background = LayerColor::create(Color4B(128,128,128,255), 960, 640);

background->setPosition(Vec2(0, 0));

background->setTag(110);

this->addChild(background);

auto cloud\_1 = Sprite::create("Chapter06/AnimationScene/animation/background/cloud.png");

cloud\_1->setPosition(Vec2(visibleSize.width/2, visibleSize.height/2));

cloud\_1->setTag(112);

this->addChild(cloud\_1);

auto cloud\_2 = Sprite::create("Chapter06/AnimationScene/animation/background/cloud.png");

cloud\_2->setPosition(Vec2(cloud\_1->getPositionX() + cloud\_1->getContentSize().width, visibleSize.height/2));

cloud\_2->setTag(113);

this->addChild(cloud\_2);

// 背景文字VS

auto tips\_vs = Label::createWithBMFont("fonts/futura-48.fnt","VS");

tips\_vs->setPosition(Vec2(visibleSize.width/2,visibleSize.height - 80));

this->addChild(tips\_vs, 5);

// 摇杆

m\_joystick = Joystick::create(Vec2(100,100), 50.0f, "Chapter06/AnimationScene/animation/control/j-btn.png", "Chapter06/AnimationScene/animation/control/j-bg.png");

this->addChild(m\_joystick, 4);

// 攻击

auto attackItem = MenuItemImage::create(

"Chapter06/AnimationScene/animation/control/j-btn.png",

"Chapter06/AnimationScene/animation/control/j-btn.png",

CC\_CALLBACK\_1(AnimationScene::attackCallback, this));

attackItem->setPosition(Vec2(visibleSize.width - 80, 100));

auto act1 = FadeOut::create(1);

auto act2 = FadeIn::create(1);

auto act3 = Sequence::create(act1, act2, NULL);

auto act = RepeatForever::create(act3); // 按钮闪烁

attackItem->runAction(act);

auto menu = Menu::create(attackItem, NULL);

menu->setPosition(Vec2::ZERO);

this->addChild(menu, 5);

// 地图

auto map = TMXTiledMap::create("Chapter06/AnimationScene/animation/background/background.tmx");

if (map == NULL)

{

CCLOG("map load failed!");

return false;

}

Size mapSize = map->getMapSize();

Size tiledSize = map->getTileSize();

map->setPosition(Vec2(0, 0));

map->setTag(111);

this->addChild(map, 2);

// enemy

m\_enemy = Enemy::create(Vec2(visibleSize.width - 200, 150));

this->addChild(m\_enemy, 3);

// hero

m\_player = Hero::create(Vec2(visibleSize.width/4, 100));

this->addChild(m\_player, 3);

// AI

m\_aimanager = AIManager::create(this);

m\_aimanager->setAI(m\_enemy, m\_player);

// 碰撞检测

m\_contactListener = MyContactListener::create(this, m\_player, m\_enemy);

// 血条

m\_bloodBar\_h = Sprite::create("Chapter06/AnimationScene/animation/background/1-new.png");

m\_bloodBar\_h->setTag(10000);

auto blood\_bg\_h = Sprite::create("Chapter06/AnimationScene/animation/background/1-new-f.png");

m\_bloodBar\_h->setPosition(Vec2(m\_bloodBar\_h->getContentSize().width/2 + 5, visibleSize.height - 50));

blood\_bg\_h->setPosition(Vec2(m\_bloodBar\_h->getContentSize().width/2 + 5, visibleSize.height - 50));

this->addChild(blood\_bg\_h, 5);

this->addChild(m\_bloodBar\_h, 5);

m\_bloodBar\_e = Sprite::create("Chapter06/AnimationScene/animation/background/2-new.png");

m\_bloodBar\_e->setTag(10001);

auto blood\_bg\_e = Sprite::create("Chapter06/AnimationScene/animation/background/2-new-f.png");

m\_bloodBar\_e->setPosition(Vec2(visibleSize.width - m\_bloodBar\_e->getContentSize().width/2 - 5, visibleSize.height - 50));

blood\_bg\_e->setPosition(Vec2(visibleSize.width - m\_bloodBar\_e->getContentSize().width/2 - 5, visibleSize.height - 50));

this->addChild(blood\_bg\_e, 5);

this->addChild(m\_bloodBar\_e, 5);

// update

this->scheduleUpdate();

return true;

}

Scene\* AnimationScene::createScene()

{

auto scene = Scene::create();

auto layer = AnimationScene::create();

scene->addChild(layer);

return scene;

}

void AnimationScene::update(float delta)

{

// 背景云动画

Size visibleSize = Director::getInstance()->getVisibleSize();

auto cloud\_1 = (Sprite\*)getChildByTag(112);

auto cloud\_2 = (Sprite\*)getChildByTag(113);

if (cloud\_1->getPositionX() > -(3 / 2 \*cloud\_1->getContentSize().width - visibleSize.width))

{

cloud\_1->setPositionX(cloud\_1->getPositionX() - 1);

} else

{

cloud\_1->setPositionX(visibleSize.width/2);

}

cloud\_2->setPositionX(cloud\_1->getPositionX() + cloud\_1->getContentSize().width);

// 更新血条

float temp\_e = float(m\_enemy->getLife()) / m\_enemy->getMaxLife();

float x = 44 + (354-88-44)\*(1-temp\_e);

m\_bloodBar\_e->setTextureRect(Rect(x,0,354 - x,m\_bloodBar\_e->getContentSize().height));

m\_bloodBar\_e->setPositionX(visibleSize.width - m\_bloodBar\_e->getContentSize().width/2 - 5);

float temp\_h = float(m\_player->getLife()) / m\_player->getMaxLife();

m\_bloodBar\_h->setTextureRect(Rect(0,0,88+(354-88-44)\* temp\_h,m\_bloodBar\_h->getContentSize().height));

m\_bloodBar\_h->setPositionX(m\_bloodBar\_h->getContentSize().width/2 + 5);

// 控制角色移动

if (m\_joystick->getDirection() == Joystick\_dir::\_RIGHT)

{

m\_player->play(MOVERIGHT);

} else if (m\_joystick->getDirection() == Joystick\_dir::\_LEFT)

{

m\_player->play(MOVELEFT);

} else if (m\_joystick->getDirection() == Joystick\_dir::\_STOP)

{

m\_player->play(STAND);

}

}

void AnimationScene::attackCallback(Ref\* pSender)

{

m\_player->play(ATTACK);

m\_enemy->play(ATTACK);

}