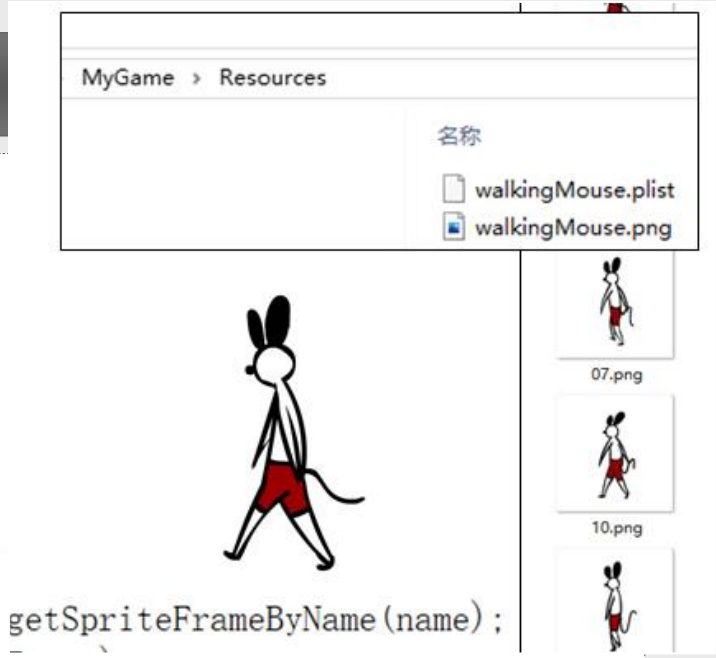
第七章 二维游戏动画合成

第八章 碰撞检测和运动模拟

上节回顾

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



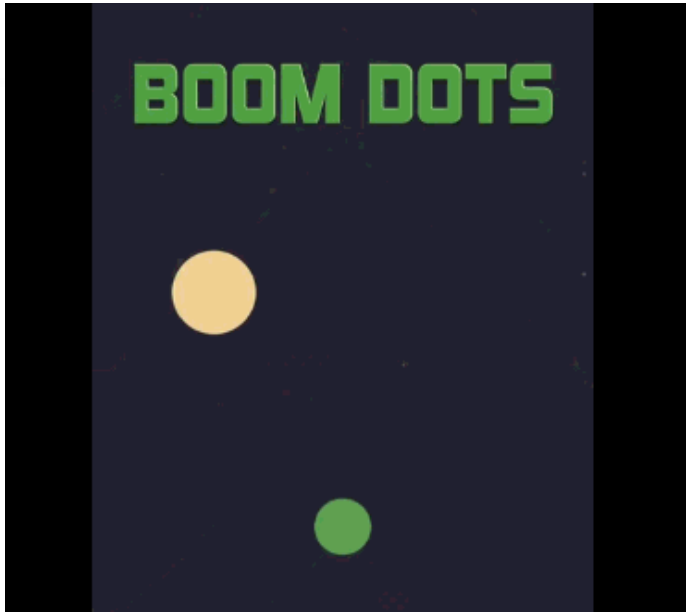
本节内容

- Chapter 7
  - 游戏动画实例----侠客行（上）
- Chapter 8
  - 碰撞检测----侠客行（下）

实验 & 作业赏析

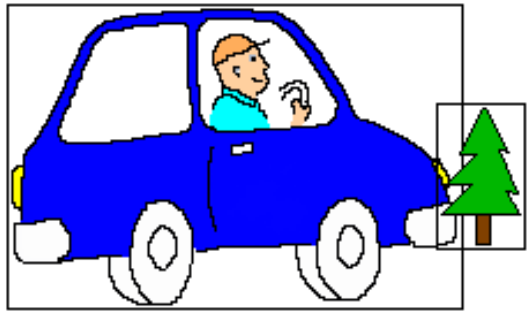
碰撞检测

- 碰撞检测
  - 对运动物体的碰撞判断是许多游戏中不可或缺的要素
    - ✓人物与物体碰撞
    - ✓人物与人物碰撞
  - 常见的碰撞检测方法：
    - (1) 区域检测
    - (2) 颜色检测
    - (3) 碰撞点检测



碰撞检测

- 区域检测
  - 适用于精确度要求不高的场合
  - 简化为矩形或圆形
  - 检测方法
    - ✓将被检测物体置于某种规则形状之中进行判断



（1）矩形：

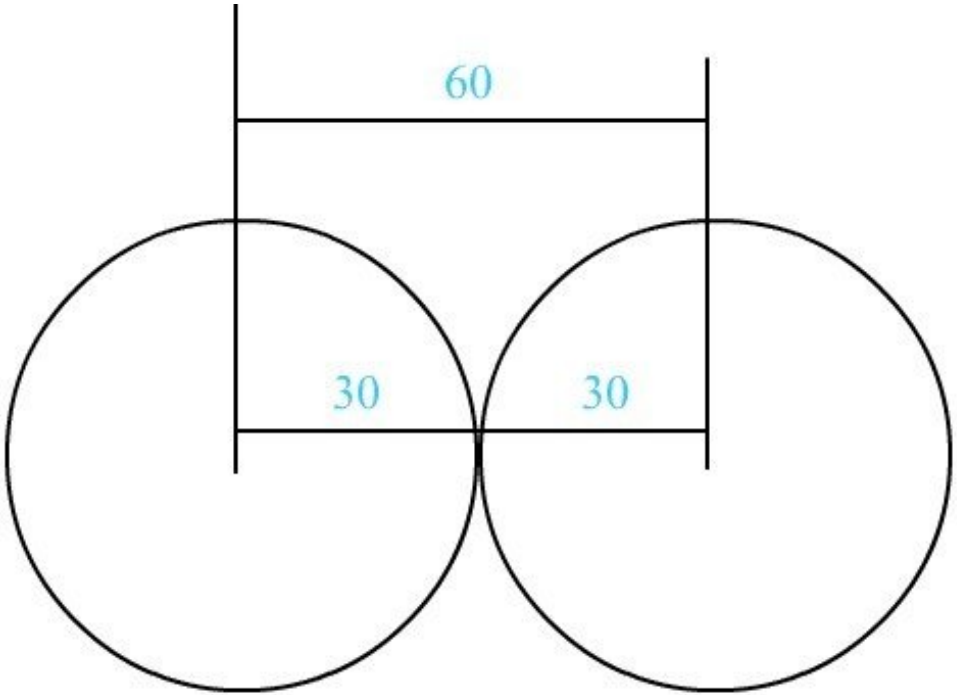
通过比较各自的左上角和右下角的坐标来检测

（2）圆：

通过圆心距与各自半径的关系来检测

碰撞检测

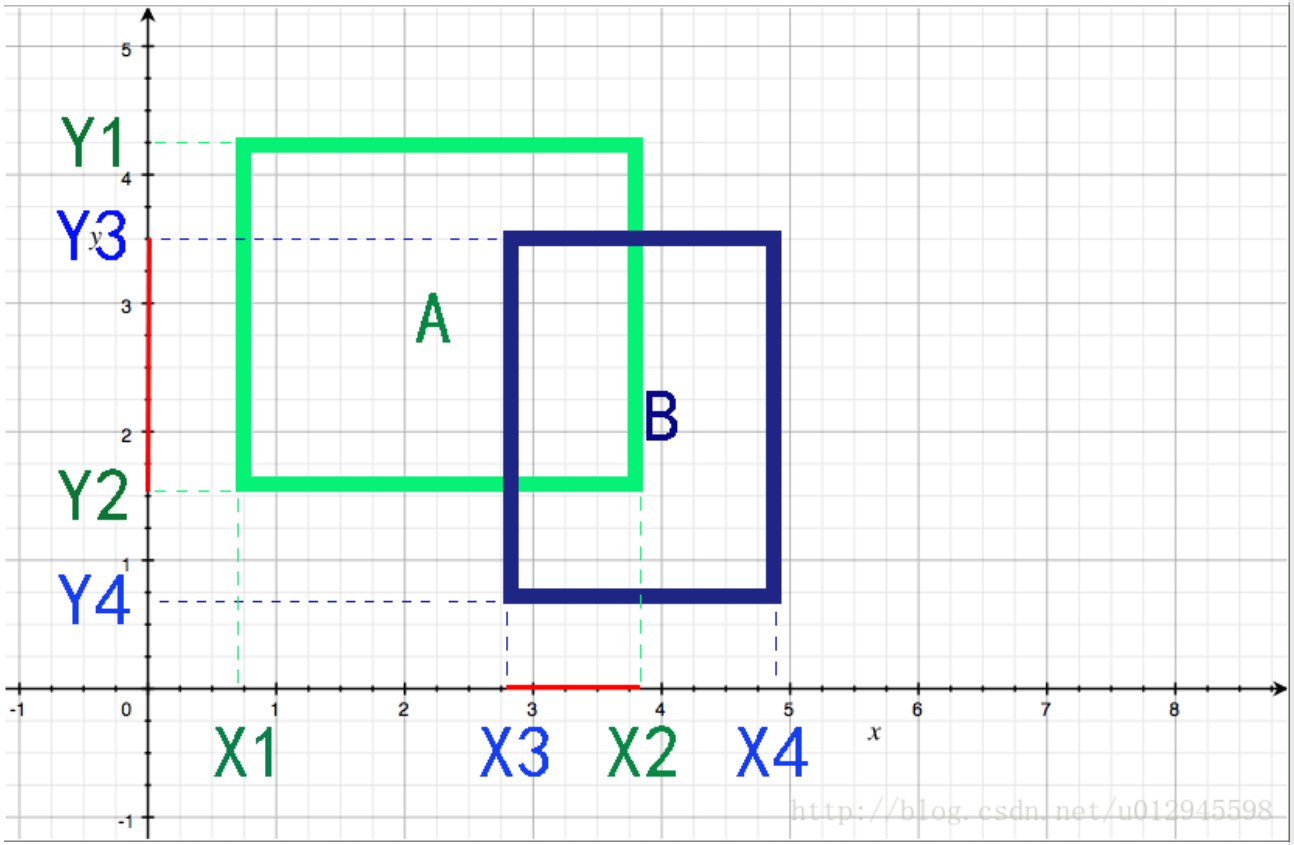
- 区域检测
  - 圆形区域



观察两个圆的碰撞，我们可以检测两个圆的距离，如果距离小于两个圆的半径，则表示碰撞上了。

碰撞检测

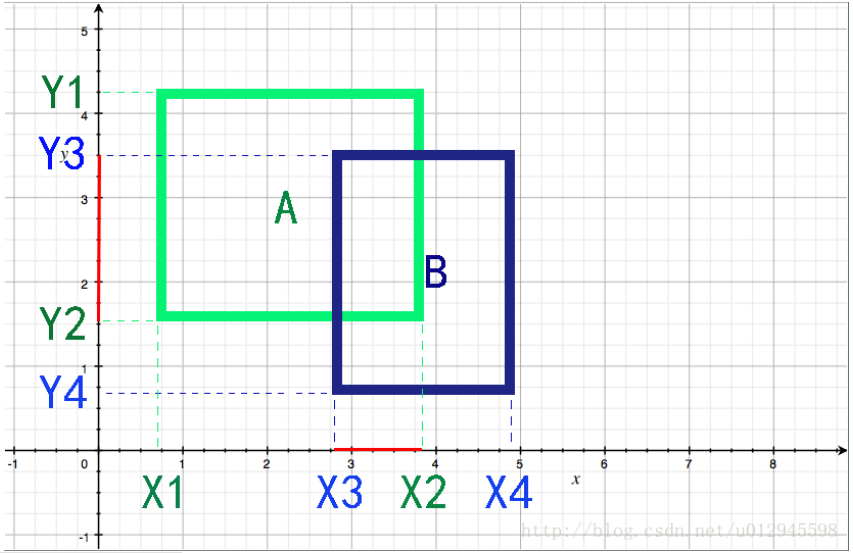
- 区域检测
  - 矩形区域



物体A与物体B分别沿两个坐标轴做投影，只有在两个坐标轴都发生重叠的情况下，两个物体才意味着发生了碰撞。

碰撞检测

- 区域检测
  - 矩形区域



（1）物体A的Y轴方向最小值大于物体B的Y轴方向最大值；

（2）物体A的X轴方向最小值大于物体B的X轴方向最大值；

（3）物体B的Y轴方向最小值大于物体A的Y轴方向最大值；

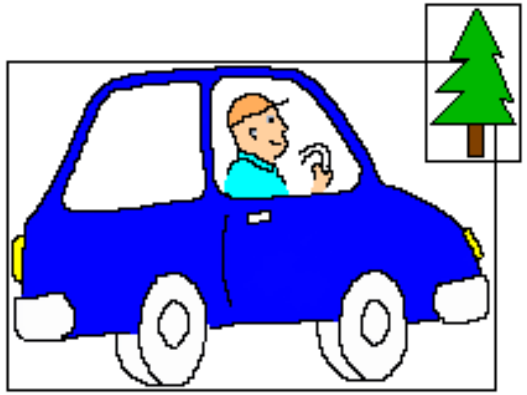
（4）物体B的X轴方向最小值大于物体A的X轴方向最大值；

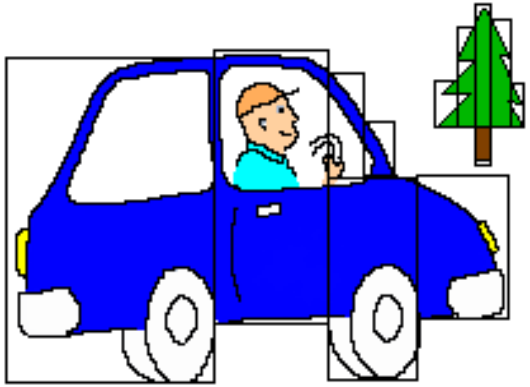
或

若满足上述条件，则证明物体A与物体B并未发生重合，反之，则证明物体A与物体B重合。

碰撞检测

- 区域检测
  - 优点
    - ✓检测速度快



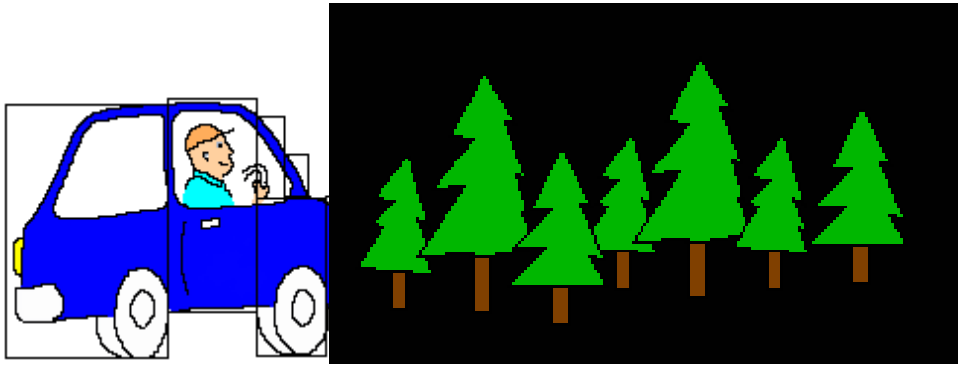


  - 缺点
    - ✓精度不高，有可能出现两物体还没碰撞就误判的情况
  - 解决方法
    - ✓对物体设定多个检测区域，尽量使检测区域的外形与物体轮廓接近

碰撞检测

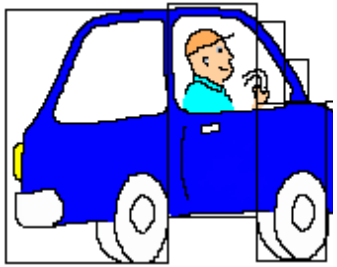
- 颜色检测
  - 优点
    - ✓较为精确的检测方式
  - 缺点
    - ✓相对耗时

为树林制作掩码图
将树林主体轮廓用黑色填充



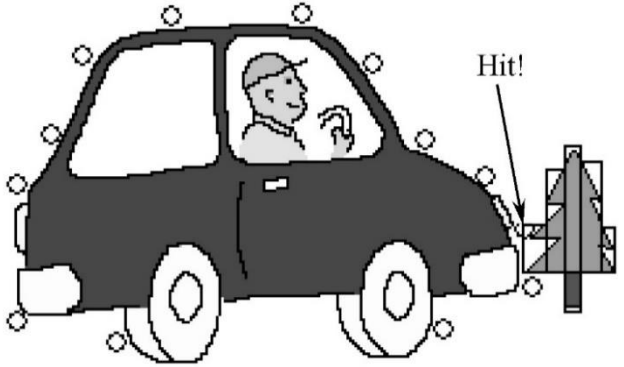
碰撞检测

- 检测汽车与树林的碰撞(遮盖)
  - 将汽车图像上的点和掩码图上相应位置的点按位“与”操作
  - 检查结果中是否有黑色点（RGB值为0）存在
  - 任何颜色的RGB值与黑色图形进行按位与运算，将得到黑色



碰撞检测

- 碰撞点检测
  - 也算区域检测的一种
  - 用碰撞点代替碰撞区域
  - 运用得当的话，可减少碰撞检测工作量
  - 检测方法：



（1）在两个运动物体之中的一个物体上设置碰撞点，在另一个物体上设置检测区域

（2）游戏运行时，逐个判断碰撞点是否在检测区域中

游戏动画实例——侠客行

- MyContactListener类
  - 继承自Node
  - 成员变量
  - 成员方法

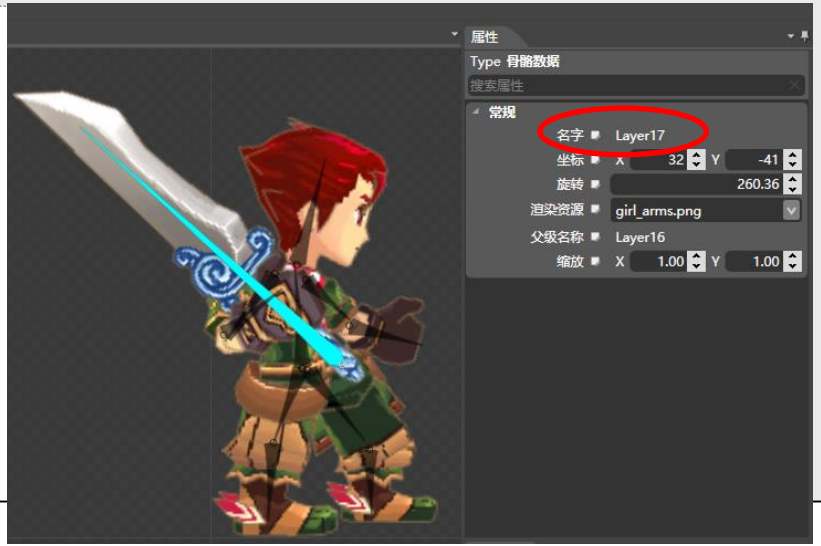
```
class MyContactListener : public Node
```

```
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

tactListener.h | Hero.h | MyContactListener.cpp | AIManager.h | Enemy.h | Enemy.cpp

(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
  - 增加防御动作（DEFENCE，以一定概率抵消武器伤害）
  - 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：6.1周二晚23:59分

目录

本节内容

- Chapter 7
  - 游戏动画实例----侠客行（上）
- Chapter 8
  - 碰撞检测----侠客行（下）

实验点评 & 期末大作业布置

实验点评 & 期末大作业布置

转场前