现代操作系统应用开发实验报告

学号：153311117　　**班级　　：**教务 2 班

姓名：黄楠绚　　**实验名称：**homework11

一．参考资料

请在这里列出对本实验有帮助你所参考的资料或者网站。

cocos2d-x action 执行完毕的回调：

http://blog.csdn.net/ubuntu64fan/article/details/43447159

cocos2dx 3.1 倒计时实现：

http://blog.csdn.net/hsljz/article/details/40862663

cocos2dx blood bar， cocos2dx 血条 最简单实现，带动画：

http://www.tuicool.com/articles/RVZnqa

二．实验步骤

HelloWorldSence.h 头文件中加入我自己定义的私有函数和变量：

```
void update(float dt);  // 调度器的update函数，用于倒计时

// 跑四个方向
enum directions { up, down, left, right } key_directions;
void up_run(cocos2d::Ref* pSender);
void down_run(cocos2d::Ref* pSender);
void left_run(cocos2d::Ref* pSender);
void right_run(cocos2d::Ref* pSender);

// 三个动画
void running(int direction);
void attacking(cocos2d::Ref* pSender);
void dying(cocos2d::Ref* pSender);

// 关于X和Y动画不能同时播放
bool triggering;        // x和y不能同时播放、精灵智能响应wasd中的一个动作的flag
void enable_trigger();  // 使triggering为真
```

demo 和 ppt 已经提供一些实例代码，模仿实例代码写下死亡动画：

```
// 死亡动画(帧数：22帧，高：90，宽：79)
auto texture2 = Director::getInstance()->getTextureCache()->addImage("$lucia_dead.png");
dead.reserve(22);
for (int i = 0; i < 22; i++) {
    auto frame = SpriteFrame::createWithTexture(texture2, CC_RECT_PIXELS_TO_POINTS(Rect(79 * i, 0, 79, 90)));
    dead.pushBack(frame);
}
```

由于 demo 的运动动画是从第 3 张图开始的，因此，
在运动动画的 createWithTexture 函数需要 i+3，并且取余 8 使得到的数字不超过 8 且形成循环：
(ps: 实验报告快写好的时候被同学告知 demo 的动画应该只使用前 3 帧，但我就没有改过来了……)

```
auto frame = SpriteFrame::createWithTexture(texture3, CC_RECT_PIXELS_TO_POINTS(Rect(68 * ((i + 3) % 8), 0, 68, 101)));
run.pushBack(frame);
```

```
// 运动动画(帧数：8帧，高：101，宽：68)
auto texture3 = Director::getInstance()->getTextureCache()->addImage("$lucia_forward.png");
run.reserve(8);
for (int i = 0; i < 8; i++) {
    auto frame = SpriteFrame::createWithTexture(texture3, CC_RECT_PIXELS_TO_POINTS(Rect(68 * ((i + 3) % 8), 0, 68, 101)));
    run.pushBack(frame);
}
```

利用 MenuItemLabel 设置左下角的虚拟方向 wasd：

```
// 设置左下角的虚拟方向wasd
auto w_diction = MenuItemLabel::create(Label::createWithTTF("W", "fonts/arial.ttf", 36), CC_CALLBACK_1(HelloWorld::up_run, this));
w_diction->setPosition(Vec2(origin.x + 1.5 * w_diction->getContentSize().width, origin.y + 2 * w_diction->getContentSize().height));
auto w_menu = Menu::create(w_diction, NULL);
w_menu->setPosition(Vec2::ZERO);
this->addChild(w_menu, 2);

auto a_diction = MenuItemLabel::create(Label::createWithTTF("A", "fonts/arial.ttf", 36), CC_CALLBACK_1(HelloWorld::left_run, this));
a_diction->setPosition(Vec2(origin.x + a_diction->getContentSize().width, origin.y + a_diction->getContentSize().height));
auto a_menu = Menu::create(a_diction, NULL);
a_menu->setPosition(Vec2::ZERO);
this->addChild(a_menu, 2);

auto s_diction = MenuItemLabel::create(Label::createWithTTF("S", "fonts/arial.ttf", 36), CC_CALLBACK_1(HelloWorld::down_run, this));
s_diction->setPosition(Vec2(origin.x + 1.5 * w_diction->getContentSize().width, origin.y + s_diction->getContentSize().height));
auto s_menu = Menu::create(s_diction, NULL);
s_menu->setPosition(Vec2::ZERO);
this->addChild(s_menu, 2);

auto d_diction = MenuItemLabel::create(Label::createWithTTF("D", "fonts/arial.ttf", 36), CC_CALLBACK_1(HelloWorld::right_run, this));
d_diction->setPosition(Vec2(origin.x + 2.5 * w_diction->getContentSize().width, origin.y + d_diction->getContentSize().height));
auto d_menu = Menu::create(d_diction, NULL);
d_menu->setPosition(Vec2::ZERO);
```

利用 MenuItemLabel 设置右下角的 xy 行为键：
先初始化 triggering：

```
// 初始化triggering为false
// 若为false则没有死亡或攻击动画进行，可以按下X或Y发生动作
// 否则说明已经有动画进行，按下X或Y不能发生响应
triggering = false;
```

设置 xy：

```
auto dead_button = MenuItemLabel::create(Label::createWithTTF("X", "fonts/arial.ttf", 36), CC_CALLBACK_1(HelloWorld::dying, this));
dead_button->setPosition(Vec2(origin.x + visibleSize.width - dead_button->getContentSize().width, origin.y + 2 * dead_button->getContentSize().height))
auto dead_menu = Menu::create(dead_button, NULL);
dead_menu->setPosition(Vec2::ZERO);
this->addChild(dead_menu, 2);

auto attack_button = MenuItemLabel::create(Label::createWithTTF("Y", "fonts/arial.ttf", 36), CC_CALLBACK_1(HelloWorld::attacking, this));
attack_button->setPosition(Vec2(origin.x + visibleSize.width - 2 * attack_button->getContentSize().width, origin.y + attack_button->getContentSize().he
auto attack_menu = Menu::create(attack_button, NULL);
attack_menu->setPosition(Vec2::ZERO);
this->addChild(attack_menu, 2);
```

设置正上方的倒计时器：

```
// 设置正上方的一个计时器
// 初始化倒计时时间为180
dtime = 180;
char *init_time = new char[5];
sprintf(init_time, "%d", dtime);
// 建立一个数字label
time = Label::createWithTTF(init_time, "fonts/arial.ttf", 36);
time->setPosition(Vec2(origin.x + visibleSize.width / 2, origin.y + visibleSize.height - time->getContentSize().height));
this->addChild(time, 2);
```

```
// 设置一个间隔时间为1s的调度器用来计时
schedule(schedule_selector(HelloWorld::update), 1.0f);
```

```
void HelloWorld::update(float dt) {
    --dtime;
    char *dtime_str = new char[5];
    sprintf(dtime_str, "%d", dtime);
    // 更新计时器label的文字
    time->setString(dtime_str);
    // 如果dtime为0，取消调度器
    if (dtime == 0) {
        unschedule(schedule_selector(HelloWorld::update));
    }
}
```

awsd 四个方向对应的回调函数：
判断是否有其他动作正在发生，如果 triggering 为 false，则把 triggering 设为 true 开始执行 running

```
void HelloWorld::up_run(cocos2d::Ref* pSender) {
    if (!triggering) {
        triggering = true;
        running(up);
    }
}

void HelloWorld::down_run(cocos2d::Ref* pSender) {
    if (!triggering) {
        triggering = true;
        running(down);
    }
}

void HelloWorld::left_run(cocos2d::Ref* pSender) {
    if (!triggering) {
        triggering = true;
        running(left);
    }
}

void HelloWorld::right_run(cocos2d::Ref* pSender) {
    if (!triggering) {
        triggering = true;
        running(right);
    }
}
```

running 函数：

生成走路动画和回调函数：

```
// 走路的动画
auto animation = Animation::createWithSpriteFrames(run, 0.03f);
auto animate = Animate::create(animation);
// 动作执行完毕后把triggering设为false
auto func = CallFuncN::create(CC_CALLBACK_0(HelloWorld::enable_trigger, this));
// 根据方向判断
```

再根据传递进来的参数方向判断，以 W（向上）方向为例子：

```
// 根据方向判断
switch (direction) {
    case up:
        // 判断移动后的位置是否出界
        // 若不出界
        if (player->getPosition().y + 30 < origin.y + visibleSize.height) {
            // 建立MoveTo移动动作
            auto move_to = MoveTo::create(0.3f, Vec2(player->getPosition().x, player->getPosition().y + 30));
            // 移动和动画同时发生
            auto my_spawn = Spawn::createWithTwoActions(animate, move_to);
            // 在my_spawn动作完成后执行回调函数
            auto seq = Sequence::create(my_spawn, func, nullptr);
            player->runAction(seq);
        }
        else {  // 若出界，则只执行动画
            auto seq = Sequence::create(animate, func, nullptr);
            player->runAction(seq);
        }
        break;
```

其他三个方向代码思路相同。

对于 X 和 Y，其中防止 X、Y 同时播放的关键代码在于：

```
// 先进行判断triggering是否为false
// 如果是false则可以播放动画
if (!triggering) {
    // 设置triggering为真，再次按下X或Y时不能同时播放动画
    triggering = true;
    // 血条减少
```

```
// 使用回调函数在动画执行完毕后
// 在动画播放完毕后，将triggering设置为false，允许按下X或Y可以播放对应动画
auto func = CallFuncN::create(CC_CALLBACK_0(HelloWorld::enalbe_trigger, this));
// 定时动画，恢复动画，静止动画，回调函数
```

加分项：使用 CCProgressFromTo 实现血条均匀增加和减少，两个代码相似：

```
// 血条减少
float progressTo = pT->getPercentage() - 20;
float progressFrom = pT->getPercentage();
CCProgressFromTo *from_to = CCProgressFromTo::create(1.5, progressFrom, progressTo);
pT->runAction(from_to);
```

```
float progressTo = pT->getPercentage() + 20;
float progressFrom = pT->getPercentage();
CCProgressFromTo *from_to = CCProgressFromTo::create(1.5, progressFrom, progressTo);
pT->runAction(from_to);
```

X 的死亡动画播放代码和 Y 的攻击动画代码相似，如下图：　死亡动画的：

```
void HelloWorld::dying(cocos2d::Ref* pSender) {
    // 先进行判断triggering是否为false
    // 如果是false则可以播放动画
    if (!triggering) {
        // 设置triggering为真，再次按下X或Y时不能同时播放动画
        triggering = true;
        // 血条均匀减少
        float progressTo = pT->getPercentage() - 20;
        float progressFrom = pT->getPercentage();
        CCProgressFromTo *from_to = CCProgressFromTo::create(1.5, progressFrom, progressTo);
        pT->runAction(from_to);
        // 死亡动画
        auto dead_animation = Animation::createWithSpriteFrames(dead, 0.07f);
        auto dead_animate = Animate::create(dead_animation);
        // 静止动画
        auto idle_animation = Animation::createWithSpriteFrames(idle, 0.01f);
        auto idle_animate = Animate::create(idle_animation);

        // 使用回调函数在动画执行完毕后
        // 在动画播放完毕后，将triggering设置为false，允许按下X或Y可以播放对应动画
        auto func = CallFuncN::create(CC_CALLBACK_0(HelloWorld::enalbe_trigger, this));
        // 序列动画：死亡动画-->静止动画-->回调函数
        auto seq = Sequence::create(dead_animate, idle_animate, func, nullptr);

        player->runAction(seq);
    }
}
```

攻击动画的：

```
void HelloWorld::attacking(cocos2d::Ref* pSender) {
    // 先进行判断triggering是否为false
    // 如果是false则可以播放动画
    if (!triggering) {
        // 设置triggering为真，再次按下X或Y时不能同时播放动画
        triggering = true;
        // 血条均匀增加
        float progressTo = pT->getPercentage() + 20;
        float progressFrom = pT->getPercentage();
        CCProgressFromTo *from_to = CCProgressFromTo::create(1.5, progressFrom, progressTo);
        pT->runAction(from_to);
        // 攻击动画
        auto attatck_animation = Animation::createWithSpriteFrames(attack, 0.07f);
        auto attatck_animate = Animate::create(attatck_animation);
        // 静止动画
        auto idle_animation = Animation::createWithSpriteFrames(idle, 0.01f);
        auto idle_animate = Animate::create(idle_animation);

        // 使用回调函数在动画执行完毕后
        // 在动画播放完毕后，将triggering设置为false，允许按下X或Y可以播放对应动画
        auto func = CallFuncN::create(CC_CALLBACK_0(HelloWorld::enalbe_trigger, this));
        // 序列动画：攻击动画-->静止动画-->回调函数
        auto seq = Sequence::create(attatck_animate, idle_animate, func, nullptr);

        player->runAction(seq);
    }
}
```

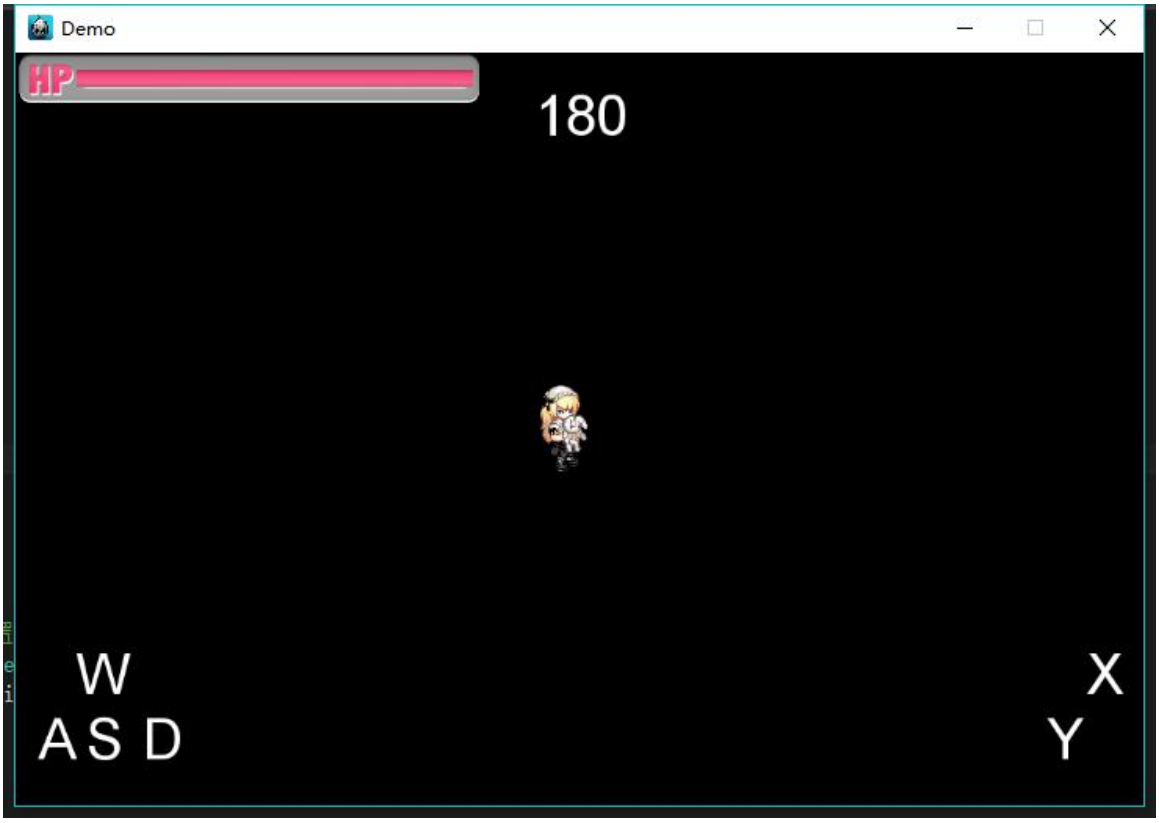
序列动作最后的回调函数代码为：

```
// 设置触发为false，允许其他动画
void HelloWorld::enalbe_trigger() {
    triggering = false;
}
```

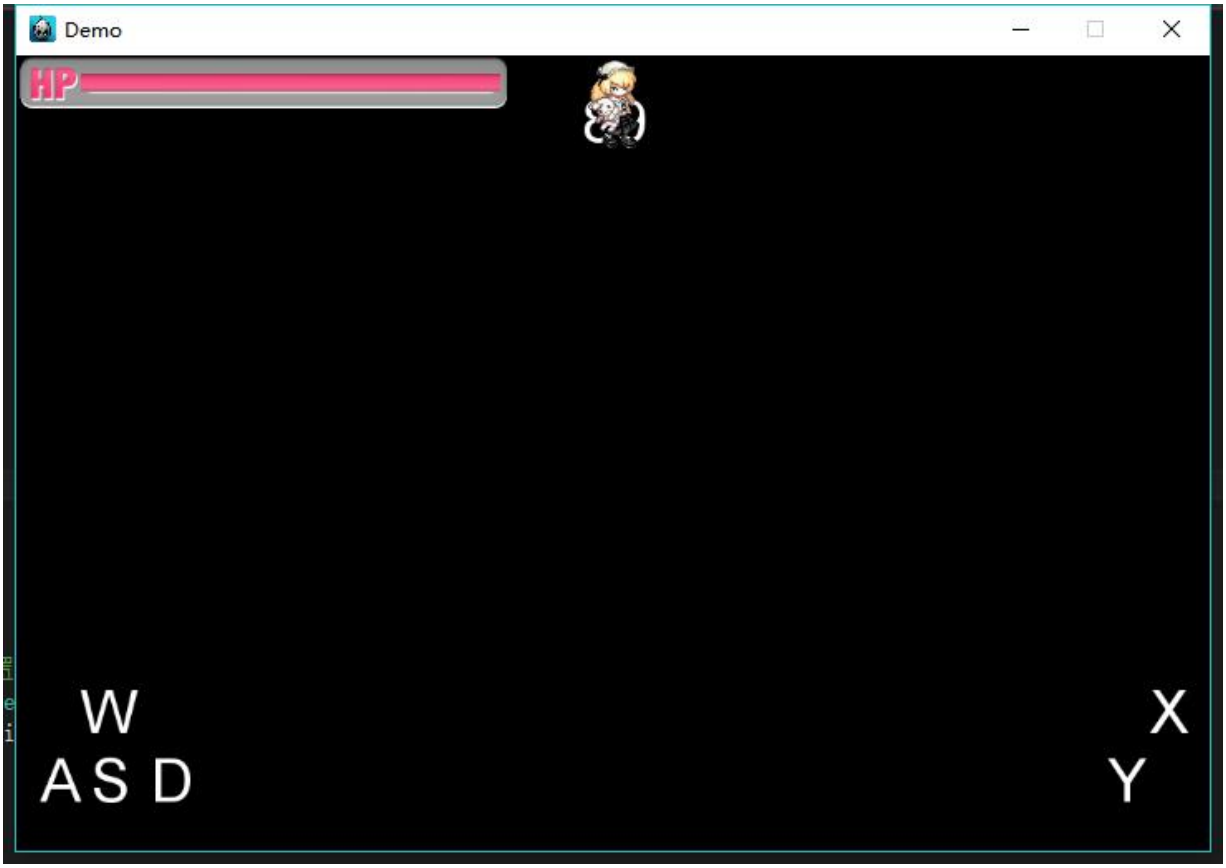
三．实验结果截图

请在这里把实验所得的运行结果截图。

运行代码得到的初始化界面：

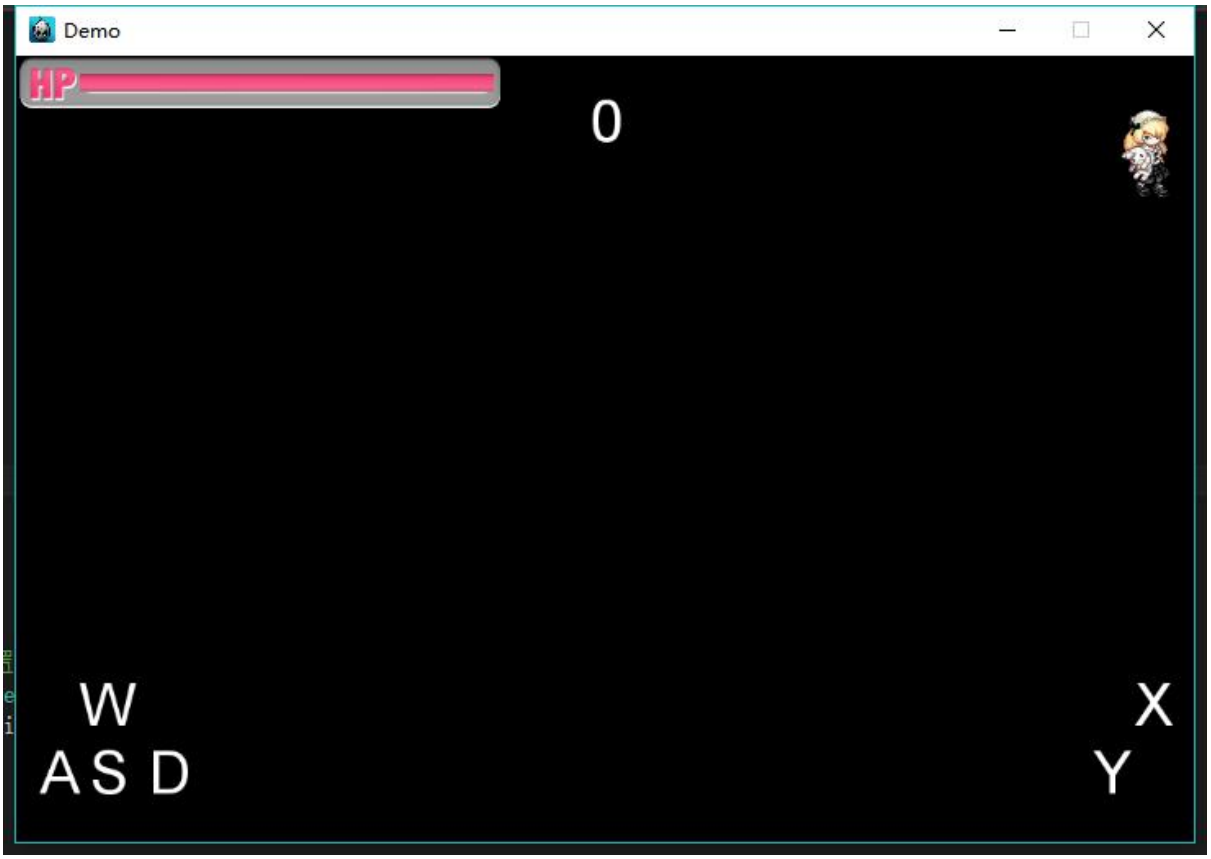


测试 W，精灵走到最上面的效果如下图，没有走出边界：

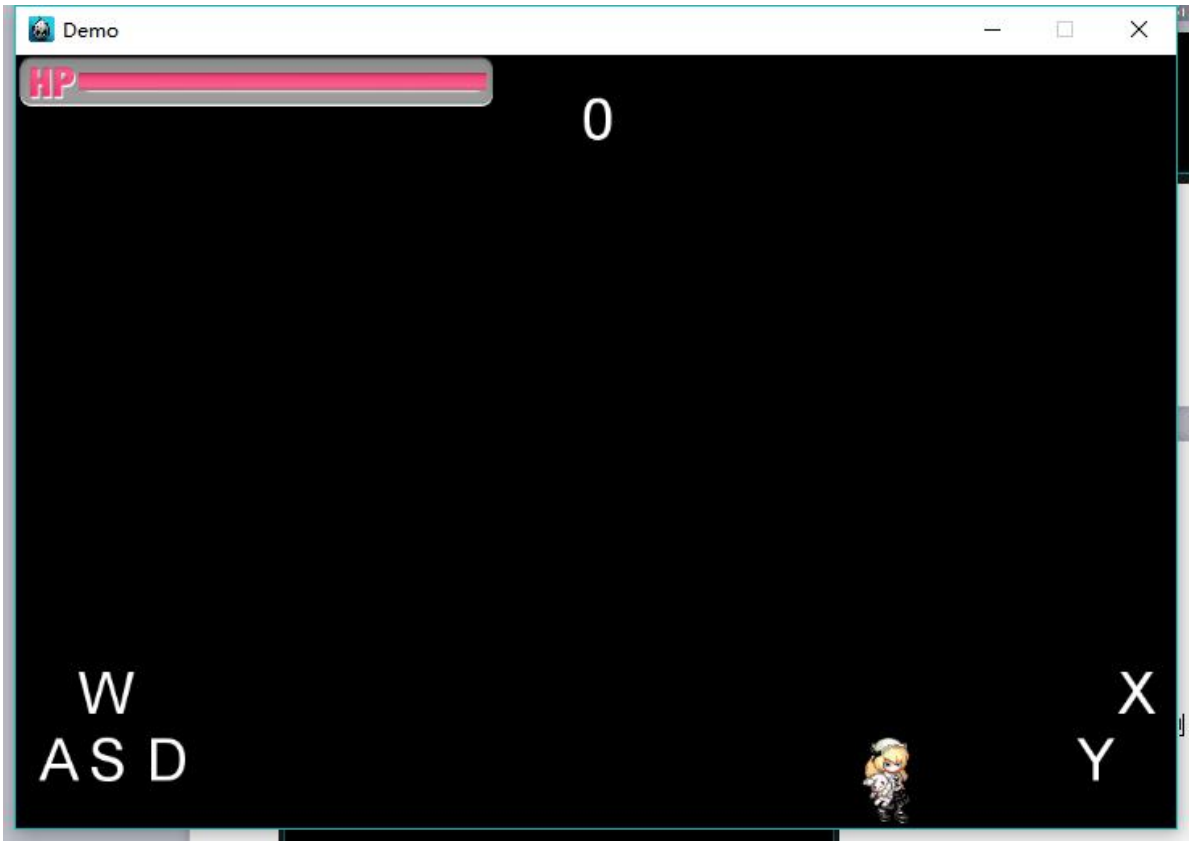


测试 A，精灵走到最左面的效果如下图，没有走出边界：

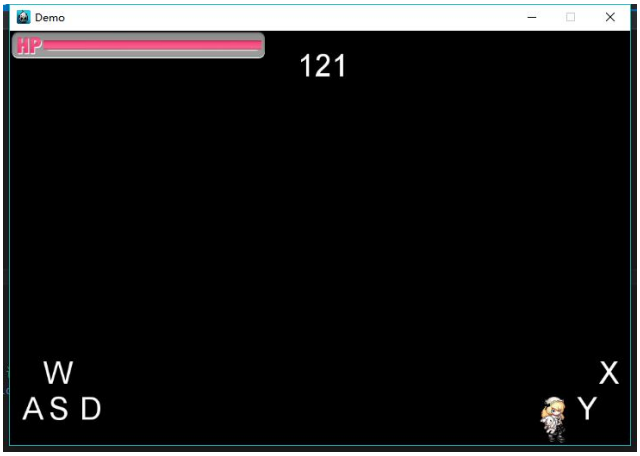
测试 D，精灵走到最右面的效果如下图，没有走出边界：

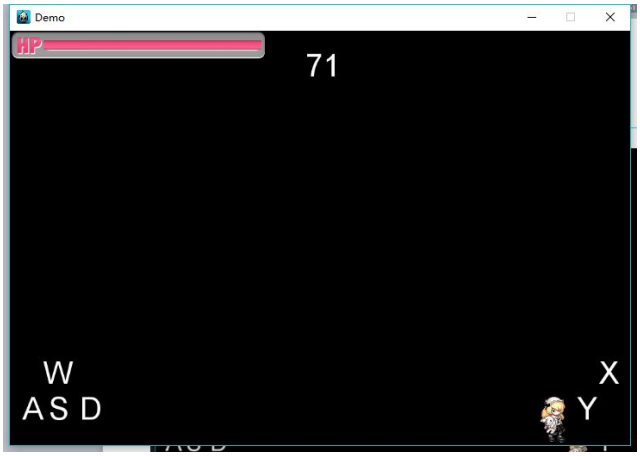


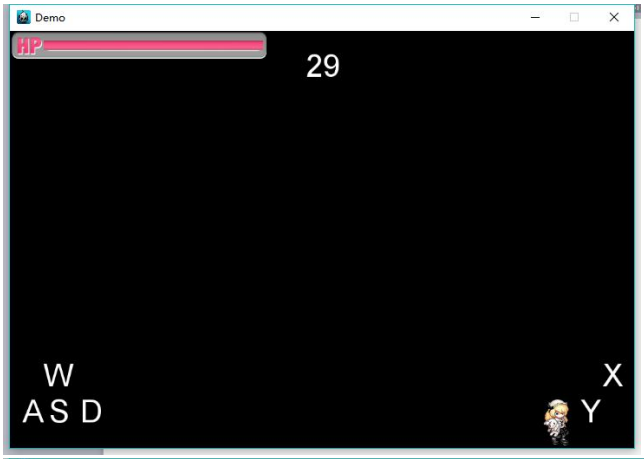
测试 S，精灵走到最下面的效果如下图，没有走出边界：

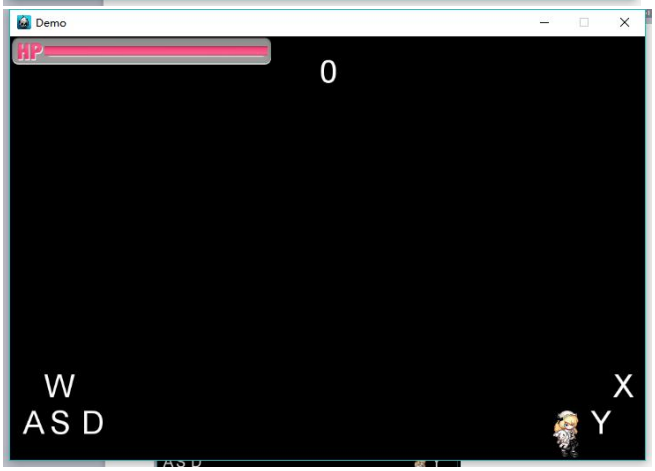


对于倒计时，从初始界面的 180 到 121 到 71 到 0 后，数字到 0 就不再发生变化，倒计时正确执行：


Demo
HP
121
W
AS D
X
Y

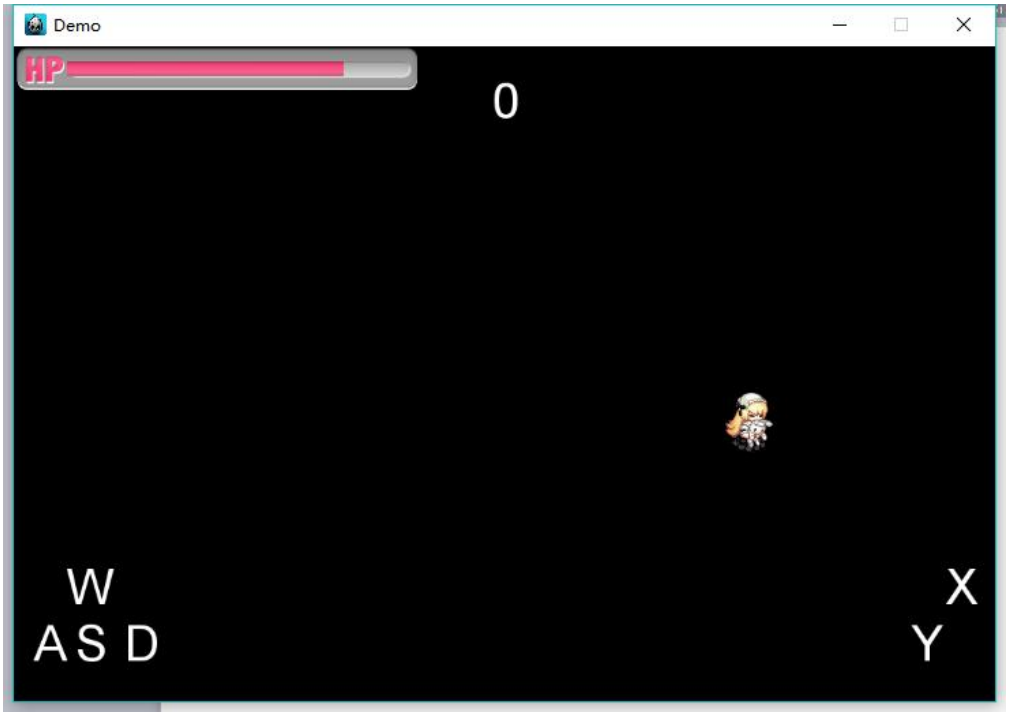

Demo
HP
71
W
AS D
X
Y


Demo
HP
29
W
AS D
X
Y

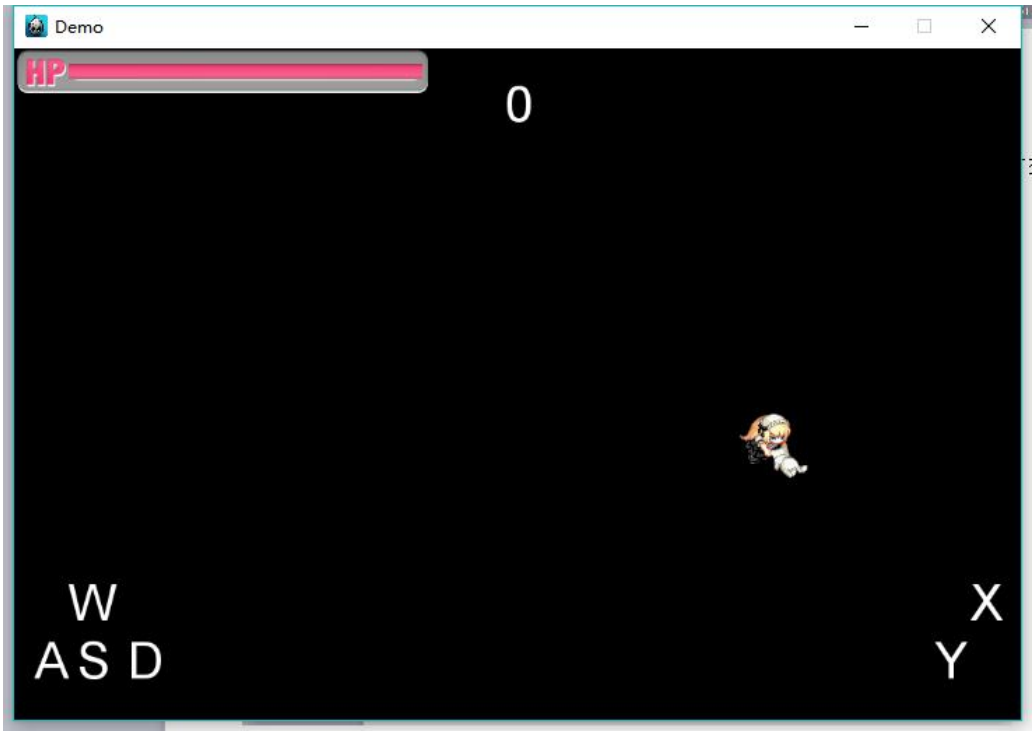

Demo
HP
0
W
AS D
X
Y

测试 X 和 Y：

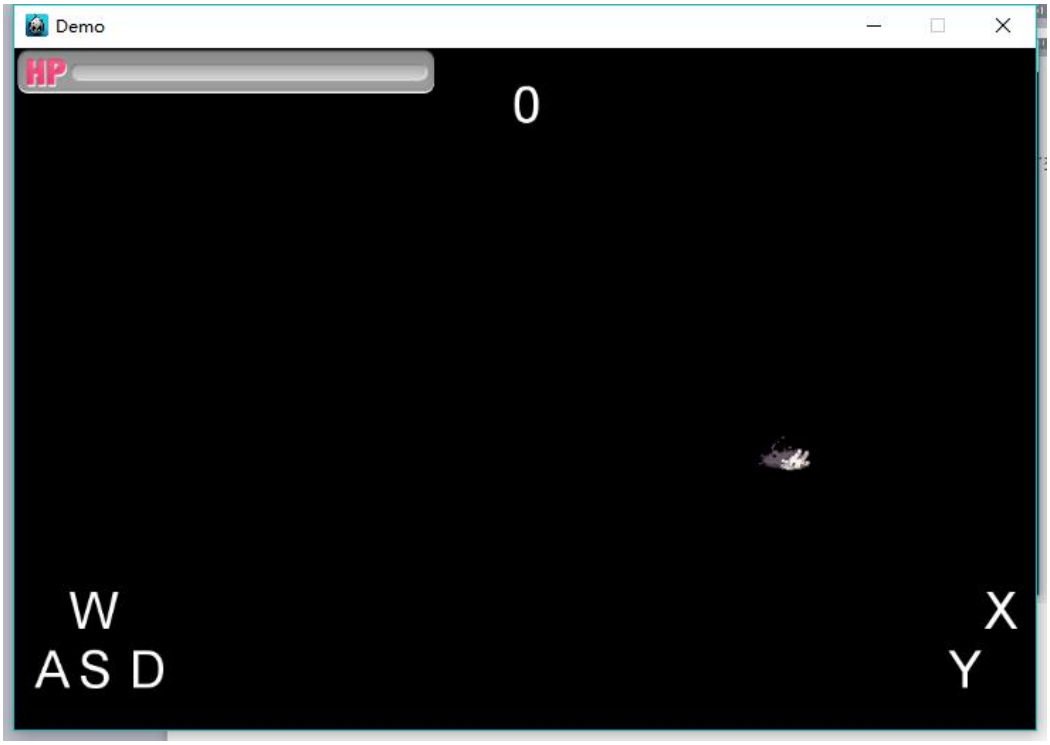
先在血条满格的状态下按下 X，出现死亡动画，血条均匀减少，快速按下 X 或 Y，不能同时播放 2 个或 2 个以上的动画：



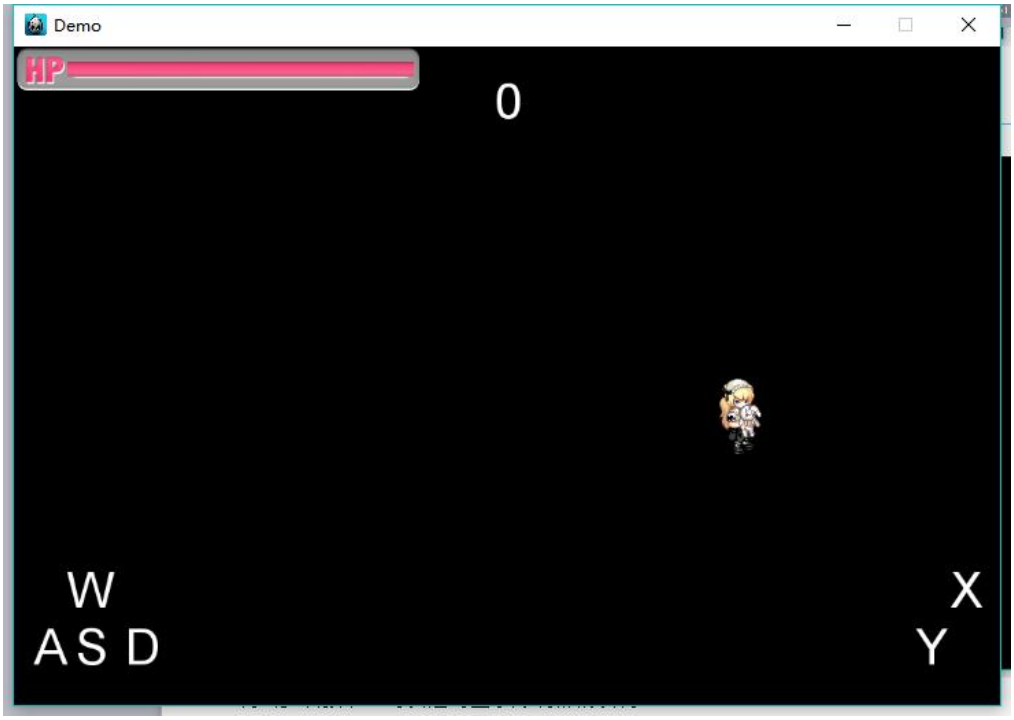
先在血条满格的状态下按下 Y，出现攻击动画，血条均匀增加，快速按下 X 或 Y，不能同时播放 2 个或 2 个以上的动画：



测试多次按下 X 可以达到血条为 0 的状况：



测试多次按下 Y 可以达到血条为满格的状况：



再测试多次随机交叉按下 X 和 Y 的情况，只能播放一种动画，测试结果无异常，此处就不放截图了

PS: 关于一些特殊情况的说明，对于我代码的运动动画，在进行测试过程中，如果鼠标点击速度适当，精灵不会走出边界，如果点击稍微快点，就可能会出现走出边界的情况。

并且，关于 ta 给的 demo，在我的电脑运行后，鼠标点击稍微快一点，虽然有时也会出现走出边界的情况，但是走出边界的频率没有我的代码高 qaq

四．实验过程遇到的问题

请在这里写下你在实验过程中遇到的问题以及解决方案。

本实验卡的第一个 bug：在创建 texture2D 动画的时候使用 createWithTexture 函数，我对这个函数的理解是对给出的连续动作长图片进行切割，一开始 copy 攻击动画的代码，下图代码中 79*i 处写成攻击动画的 113*i，一开始动画是错了，还以为是帧数间隔问题，后来仔细观察动画后才明白是把图片割错了

```
for (int i = 0; i < 22; i++) {
    auto frame = SpriteFrame::createWithTexture(texture2, CC_RECT_PIXELS_TO_POINTS(Rect(79 * i, 0, 79, 90)));
    dead.pushBack(frame);
```

五．思考与总结

请在这里写下你本次试验的心得体会以及所思所想。

这次实验相对比较简单，有些代码比如取消自定义调度函数怎么写，查了一下资料，只有如何写自定义调度 schedule 函数，没有取消的实例代码，于是根据 vs 的提示 unschedule 函数的具体参数等信息，写出 unschedule 取消自定义调度。

vs 作为一个强大的、占硬盘的 IDE，提示和报错大部分是有用的，善于利用 vs 的代码提示和报错提示，可以加快编程速度和准确度。在上周和别的同学做项目的过程中发现 vs 的报错可能没有受到足够的重视，很多时候靠 vs 的中断报错就可以解决代码问题，但如果没有认真对待中断报错把它当玄学现象，那只能使用调试工具（可能那个 bug 没有必要用调试工具）不过善于使用 IDE 自带的调试工具是程序员必备的技能之一。

1. 实验报告提交格式为 pdf。
2. 实验内容不允许抄袭，我们要进行代码相似度对比。如发现抄袭，按 0 分处理。