**我们的游戏开发日志**

组长：谢志诚

组员：吴昊、王开奇、王欲卿

**写在前面的话：**

在正式开始写这篇论文之前呢，有些话想先写下来。其实在这次实训课之前，我有一直希望能有机会的话接触一下游戏开放方面的软件，代码。这当然主要是个人兴趣，这次实训课正好给了我一个机会，所以当时我就和我的另外两个室友提出我们能不能试着去开发一下游戏，他们对此也有兴趣所以一拍即合。隔壁的小欲（王欲卿）听说之后对此也非常感兴趣，也加入了我们，所以有了我们这个四人的团队。当然我们虽然美其名曰：游戏开发，实际上并没有那么高大上，更多的可以说是四个对游戏开发一窍不通的大学生一路摸爬滚打不断尝试接触游戏开发的一个过程。整个开发历时十周，说是说十周但实际上我们每周也就是花周一下午的时间聚在一起做游戏所以严格来说是十天，在这个摸索的过程中呢，我们学习到了一定的游戏开发基础，最后也是做出了，啊，所谓的“游戏”。但是受限于我们的美工技术和时间以及试用版的种种限制，这个成品的质量真的是不敢恭维，希望老师能做好心理准备哈哈哈。哦对了，在这里也是放出我们github的网页https://github.com/siiiiil/game，老师有兴趣的话可以去那里看到我们每一次更新的时间和更新的内容以及最后的成品。

那么接下去就让我来介绍一下我们整个“游戏”的开发过程吧。

第一周

第一周的时候我们主要做了一下准备工作，首先我们四个组员聚在一起讨论了一下我们究竟要做什么类型的游戏，以及我们使用哪款引擎来编写我们的游戏。根据我们的经验游戏的引擎主要有两大类：3D引擎和2D引擎。现在市面上主流的3D引擎有：虚幻3,4、unity3D等。2D引擎主要有：unity3D的衍生产物unity2D、gamemakerstudio、RPGmaker

等。考虑到这是我们自己制作的游戏，我们不想所有素材都搬运别人的，至少要有一些我们自己制作的角色，而我们又完全不熟悉3D建模，我们最终放弃了做3D游戏的想法。而在这些2D引擎中，RPGmaker完全是傻瓜向的软件，可以说代码的部分捉襟见肘，作为实训课来说未免不太合适。因此我们最初选择unity2d作为我们的引擎来进行开发。游戏类型我们也是准备做比较简单的2d横版的过关类游戏。这样一来工作量不是很大，作为初学者的我们应该也能handle。

第二周

定好了大致的方向并且在第一周的时候安装好了软件，我们就准备开始正式编写代码了。不过在开始之前有一个问题，那就是我们需要搭建一个属于我们四个人的工作平台，这样大家就可以各自处理自己的部分最后进行整合。这个时候比较熟悉shell语言的小欲就提出了使用github。使用github还有一个非常重要的好处就是可以非常方便地进行版本管理，在之后的开发中回滚版本也是给我们带来了非常大的好处。当然，对于当时的我们，还有一个非常大的坎就是要熟悉git相关的shell语句。在简单地了解了一下github相关的语句之后我们就正式开始使用unity2d进行开发了。但是！开发过程并不顺利，这并不是说unity2d是一款不好的引擎，相反，而是因为它太好了，可以说unity2d作为一款引擎实在是太优秀了，它的核心代码有非常多的参数以及模组可以进行大量参数的改动，可以说unity2d拥有非常大的可能性。但对于我们四只刚出生的雏鸟来说，大量的数据和自由的编辑，以及它背后大量的看不懂的英文词汇反而让我们不知道从哪里入手，开发一下子就进入了瓶颈。虽然网上有一些教学视频，但是这些教学视频少则100来集，多则500多集。大量我们不需要使用的参数和资源的介绍让我们无力看完所有的教学。再者，因为unity2d实际上是unity3d的衍生产物，所以它本质上还是一款3d的引擎，只不过它隐藏了一个z轴，所以如果对unity3d不熟悉，在开发2d游戏的时候也是非常困难。于是，在苦苦挣扎了一周以后……

第三周

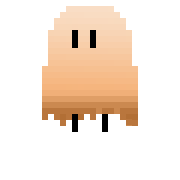
这一周我们做出了一个艰难的决定：我们要更换引擎。是的，不得不承认，unity2d是我们四个菜鸟暂时无法掌握的可怕对手，与其说是在这款引擎上苦苦挣扎不如抓紧时间找别的出路，毕竟我们只有十几周的时间。于是我们开始各自物色更适合我们的游戏引擎，这个适合这款引擎映入了我们的眼帘，那就是gamemakerstudio，简称GMS。这款GMS也是开发过著名的spelunky和hyper light drifter的游戏引擎，并且简单的看了一下它的界面，非常的简洁，代码部分也是难易度适中，它有面对傻瓜和开发者的两个版本，傻瓜版本代码部分较少，大多为模块的拖动和判定的添加。而开发者版本代码大部分需要我们自己进行编写，可以说非常适合作为学生的我们，就这样我们终于确定下来使用GMS进行开发。

第四周

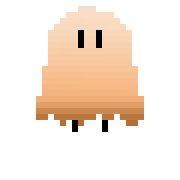
经过了三周的摸爬滚打，我们总算是可以安心下来进行游戏开发的作业了，真是的不容易。首先，我们对如何导入素材进行了研究。GMS这款引擎的核心是sprite和object的联动，sprite是指精灵，导入时导入某个物体的外貌，动作等等。Object则是指具体的某个角色或物体。Sprite好比是一个外壳，一张皮，而object则是内部的核心，具体的代码都在object里。因此，再编写角色的代码之前，我们先要给它准备一个sprite。于是我们因为没有美工，不得不自己设计了一个角色作为主角，为它做了简单的站立动画**spr\_hero\_std**，并且将它导入到了游戏之中。我们还为主角对应的obj\_hero添加了移动的判定。在step函数中加入了如下代码：

if(keyboard\_check(ord('A')))

{



x = x - 4;



if(keyboard\_check(ord('W')))

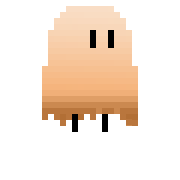
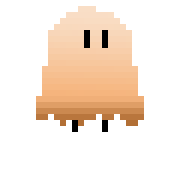
{

y = y - 4;

}

else if(keyboard\_check(ord('S')))

{



y = y + 4;

}

}

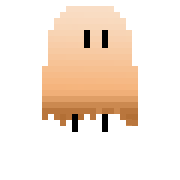
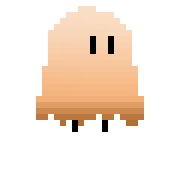
else if(keyboard\_check(ord('D')))

{

x = x + 4;

if(keyboard\_check(ord('W')))

{



y = y - 4;

}

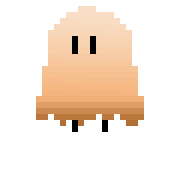
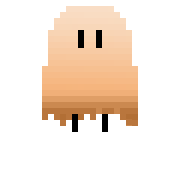
else if(keyboard\_check(ord('S')))

{

y = y + 4;

}

}



else if(keyboard\_check(ord('W')))

{

y = y - 4;

}

else if(keyboard\_check(ord('S')))

{

y = y + 4;

}

这样就完成了简单的八方向的移动。

第五周

在添加了简单的移动代码之后我们的人物就可以进行移动了，但是却没有对应的移动动画，因此第五周我们加入了两个新的sprite分别是spr\_hero\_mov\_side和spr\_hero\_mov\_back，并且在完善了移动的代码，使得在移动的时候人物会播放对应的移动的动画。同时我们还启用了主角的物理碰撞熟悉，这样一来主角就可以与场景中的物体进行互动了，那么对应的x要变成phy\_position\_x。改变后的step代码如下

if keyboard\_check(ord("A"))

{

phy\_position\_x += -8; //phy\_position\_x在打开room physics时使用

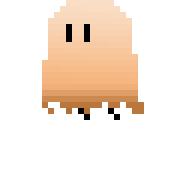
sprite\_index=spr\_hero\_mov\_side; //sprite

image\_xscale=1; //sprite方向

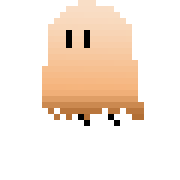
if keyboard\_check(ord("W"))

{

phy\_position\_y += -8;



sprite\_index=spr\_hero\_mov\_back;



}

if keyboard\_check(ord("S"))

{

phy\_position\_y += 8;

}

}

else if keyboard\_check(ord("D"))

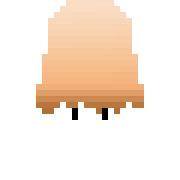
{

phy\_position\_x += 8;

sprite\_index=spr\_hero\_mov\_side;

image\_xscale=-1;

if keyboard\_check(ord("W"))



{

phy\_position\_y += -8;

sprite\_index=spr\_hero\_mov\_back;

}

if keyboard\_check(ord("S"))

{

phy\_position\_y += 8;

sprite\_index=spr\_hero\_mov\_side;

}

}

else if keyboard\_check(ord("W"))

{

//sprite\_index=;

phy\_position\_y += -8;

sprite\_index=spr\_hero\_mov\_back;

}

else if keyboard\_check(ord("S"))

{

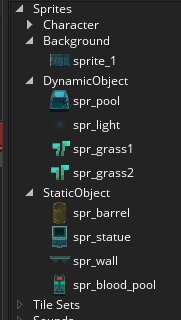
//sprite\_index=;

phy\_position\_y += 8;

sprite\_index=spr\_hero\_std;

}

第六周

这一周我们的主要任务是做主角和其他物品的物理碰撞互动，并且创建一个场景来测试这些互动，由于我们美术功底有限，背景和物体的素材要我们自己画实在是有点强人所难，所以这里就借用了一下网上的素材，从网上我们找到了8个物体和1个背景供我们测试。创建了他们对应的sprite：spr\_pool、spr\_light、spr\_grass1、spr\_grass2、spr\_barrel、spr\_statue、 spr\_wall、spr\_blood\_pool

以及object（这样我们才能做出他们的物理碰撞）：obj\_pool、obj\_light、obj\_grass1、obj\_grass2、obj\_barrel、obj\_statue、obj\_wall、obj\_blood\_pool

为这些obj添加了物理属性，并创建了一个测试空间rm\_test之后，我们发现他们可以进行碰撞了，但同时也带来了3个问题：

1. 遮蔽问题，物体与物体之间的遮蔽非常有问题，主角明明走到了桶的前面但是桶却依旧在上层
2. 翻车问题，主角在与桶发生碰撞之后会很搞笑的进行旋转，就像翻车一样，这是我们不想看到的
3. 固定位置问题，有些物体，比如说一个柱子，我们希望它能固定在一个位置，而不是被主角推着到处走。

为了解决这些问题，我们在网上找了不少资料，最终一一解决了他们。

对应第一个问题，解决方法是：在每一个有物理属性的obj的step函数中添加以下代码：

depth = -y; //即深度值越小，离相机的距离越近

第二个问题的解决方法是，在每一个有物理属性的obj的create函数中添加以下代码：

phy\_fixed\_rotation=1; //即锁定翻转角度

第三个问题的解决方法是，将这些需要固定位置的物体的create函数中添加以下代码：

density=0; //即重量为无穷

这样一来主角与这些场景中物体的互动就基本做好了，在测试中我们可以控制主角推着一个木桶到处跑，同时撞到墙上或是柱子上时会被拦住。

第七周~第九周

场景互动基本完成之后，我们就开始着手添加敌人了，那几周正好遇到连续的放假，我就要求每个组员一人设计一个敌人，并且负责制作它的动画以及AI。再经过几周的设计之后呢，大家设计的怪物也都基本成型了，那么下面就让他们各自分享一下制作的心得。

吴昊的enermy1

我们游戏一开始定的主题是僵尸大战机器人（虽然主角换了），组长要求每个组员创建一个怪物，我的第一反应是类似于机械爬虫，行动是四角爬行，攻击方式是射出激光。所以我花了几天的时间，将我的所有所需的攻击动画，行走动画，苏醒动画，起始动画完成，并导入到GameMaker中。

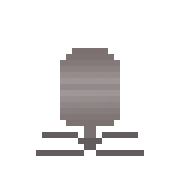
代码方面，所有的行走，即跟随主角代码，激光，主要为与其他物体碰撞方面等，都与同组组员的怪物代码差不多，但我的设计理念是当主角距离怪物一定范围之内，怪物才会开始攻击主角。这实现方法比较容易，使用point\_distance()函数，但在调试过程中，我遇到了问题，苏醒动画没有播放就直接进入行走跟随动画。因此我模仿攻击动画，加入参数m\_isleep和m\_isawake来控制动画完整播放后才进入下一个跟随状态。在攻击时，我原打算使用Alarm功能来让怪物间隔发射激光，但发现Alarm创立在create才有用，但一旦进入step，就实现不了了。我随后听从组长的建议，通过判定激光的instance是否存在来控制怪物的攻击。

这次的怪物设计还是让我受益匪浅的，虽然遇到了不少的困难，但能完成还是很欣慰的。

Create

m\_playerDirection = PlayerDirection.DOWN;

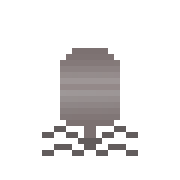
if(m\_isleep)



{image\_speed=0;

if (point\_distance(x,y,obj\_hero.x,obj\_hero.y) < 550){

m\_isleep=false;

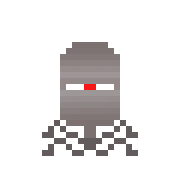


sprite\_index=spr\_enemy1\_awake;

image\_speed=1.5;

}}

Step



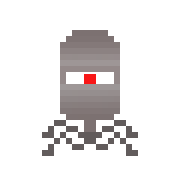
if(!m\_isAttacking &&!m\_isleep){

if(!m\_isawake)

{image\_speed=1.5;

sprite\_index = spr\_enemy1\_awake;}

else{image\_speed=1.5;



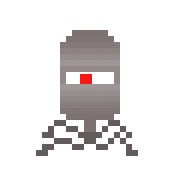
var player = instance\_find(obj\_hero, 0);

var deltaX = player.x - x;

var deltaY = player.y - y;

var mySpeed = 2;

if(deltaX > mySpeed){

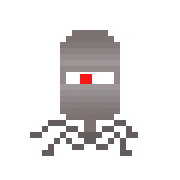


phy\_position\_x += mySpeed;

}

else if(deltaX < -mySpeed){

phy\_position\_x -= mySpeed;



}

else{

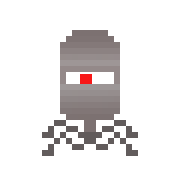
phy\_position\_x += deltaX;

}

if(deltaY > mySpeed){

phy\_position\_y += mySpeed;

}



else if(deltaY < -mySpeed){

phy\_position\_y -= mySpeed;

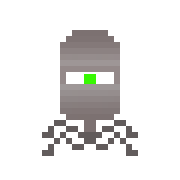
}

else{

phy\_position\_y += deltaY;

}

if(deltaX > 0){



image\_xscale = -1;

}

else if(deltaX < 0){

image\_xscale = 1;

}

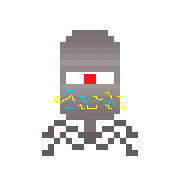
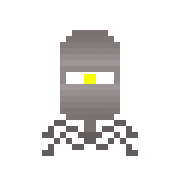
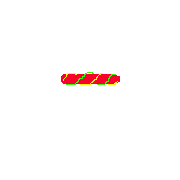
sprite\_index = spr\_enemy1\_walk;

if(m\_laser=false)

{instance\_create\_depth(x, y,-1, obj\_enemy\_1\_laser);}

}

}



phy\_fixed\_rotation=1;

m\_isAttacking = false;

m\_isleep=true;

m\_isawake=false;

m\_laser=false;

animation end

if(m\_isAttacking && sprite\_index == spr\_enemy1\_attack)

{

m\_isAttacking = false;

}

if(sprite\_index ==spr\_enemy1\_awake)

{image\_speed=0;m\_isawake=true;

}

if( sprite\_index == spr\_enemy1\_awake)

{m\_isAttacking = true;

instance\_create\_depth(x, y,-1, obj\_enemy\_1\_laser);

m\_laser=true;

m\_isAttacking = false;}

王禹卿的enermy2

我最初想要设计的是一个近战形的怪物，当主角进入范围时就会开始追击主角，与主角发生碰撞时就会造成伤害。在实际制作的过程中，最困扰的就是如何设计怪物的外观和移动的方式，一开始我做了一个类似吃豆人的模型，嘴巴一张一合去啃咬别人，移动方式是用类似一蹦一蹦的样子，在触碰地面时会有一个压扁了一样的效果，最初做了四帧的动画，但实际放入游戏中时，发现与整个游戏的风格十分的不符合，很突兀，主要是形体和色调上都很不搭配，所以被推翻了重做。通过这次的失败经验，我又进行了很多尝试，最后的结果就是一个机械鳄鱼，嘴巴一张一合，通过啃咬的方式进行攻击，这次总体来看至少能融入游戏之中了。在代码方面，我做的怪物并没有太多复杂的地方，主要是设置一个基础的AI，让怪物能自己动起来，比如说：

var player = instance\_find(obj\_hero, 0);//用来获取主角的位置

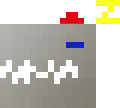
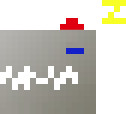
var deltaX = player.x - x;

var deltaY = player.y - y;//得到怪物与主角的距离

然后就是根据主角的位置进行移动的判定：

var mySpeed = 2;//设定移动速度

if(deltaX > mySpeed){



phy\_position\_x += mySpeed;

}

else if(deltaX < -mySpeed){

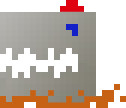
phy\_position\_x -= mySpeed;

}

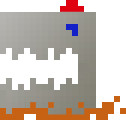
else{

phy\_position\_x += deltaX;

}

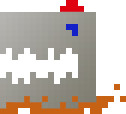


if(deltaY > mySpeed){



phy\_position\_y += mySpeed;

}



else if(deltaY < -mySpeed){

phy\_position\_y -= mySpeed;

}

else{

phy\_position\_y += deltaY;

}

之后还要注意人物方向等细节，往不同方向追击时的图像方向是相反的：

if(deltaX > 0){

image\_xscale = -1;

}

else if(deltaX < 0){

image\_xscale = 1;

}

编写在发生碰撞造成伤害时的判定：

if((m\_attachedParent.object\_index == obj\_hero && other.object\_index == obj\_enemy2)|| (m\_attachedParent.object\_index == obj\_enemy2 && other.object\_index == obj\_hero\_magic\_bullet)){

with(m\_attachedParent)

{

CharacterOnDamage();

}}

这次制作的过程虽然有过一次失败的设计，但是整体还是十分有收获的。

王开奇的enermy3

在开始设计之初我想要一个圆形的怪物，但是一直纠结于该怪物的攻击形式，然后队友给了我提示，就是采用自爆的形式来攻击，也就是说这是一个一次性的怪物，生命周期很短的怪物。在设计过程中，3号怪物朝着手榴弹的样子进行变化，绘画的过程中否定了很多的样本，最后确定了一个绿色的菠萝型手雷。其移动方式是类似于陀螺一样，可以保证其稳定性，采用上下分离的自爆形式同时应要求还给它一个醒来的过程。这样3号怪物的AI为：在主角进入到一介范围后，怪物醒来然后接近主角，当与主角一定距离后，突然冲刺接近紧接着爆炸。下面就是开始写代码了，在GameMakeStudio中，代码是完成一个事件，首先创立两个事件为create和step事件，在create中给step代码里用到的参量一个初始值

phy\_fixed\_rotation=1; //固定物体角度，使发生碰撞时不旋转

m\_issleep = true; //物体处于睡眠状态

hp=1; //物体血量为1

对于step中的代码则完成怪物的AI设计

image\_speed=1.5; //动画播放速度

depth=-y; //即深度值越小，离相机的距离越近

var player = instance\_find(obj\_hero, 0); //获取主角位置

var deltaX = player.x - x; //获取距离

var deltaY = player.y - y;

if(distance\_to\_point(player.x, player.y)<=300&&m\_issleep == true) //醒来

{ sprite\_index = spr\_enemy3\_wake;

}

if(m\_issleep==false&&distance\_to\_point(player.x, player.y)>=80) //逐渐接近

{

var mySpeed = 5;

if(deltaX > mySpeed){

phy\_position\_x += mySpeed;

}

else if(deltaX < -mySpeed){

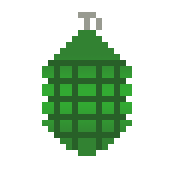
phy\_position\_x -= mySpeed;

}

else{

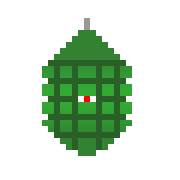
phy\_position\_x += deltaX;

}



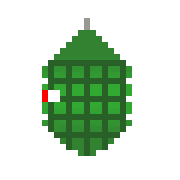
if(deltaY > mySpeed){

phy\_position\_y += mySpeed;



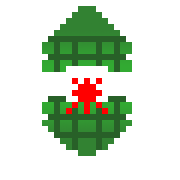
}

else if(deltaY < -mySpeed){



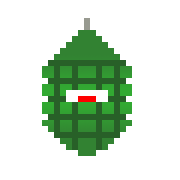
phy\_position\_y -= mySpeed;

}



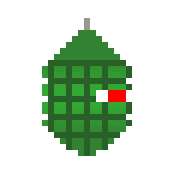
else{

phy\_position\_y += deltaY;



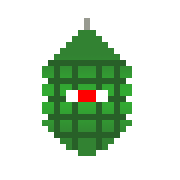
}

if(deltaX > 0){



image\_xscale = -1;

}



else if(deltaX < 0){

image\_xscale = 1;

}

sprite\_index = spr\_enemy3\_mov;

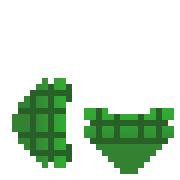
}

if(m\_issleep==false&&distance\_to\_point(player.x, player.y)<80)//突然接近

{

dashspeedx=player.x - x;

dashspeedy=player.y - y;



phy\_position\_x += dashspeedx;

phy\_position\_y += dashspeedy;

}

由于醒来只执行一次再添加一个事件Animation end

if(m\_issleep&&sprite\_index == spr\_enemy3\_wake)

{

m\_issleep=false;

}

也就是当醒来动画播放后，参量m\_issleep变为false，不符合醒来播放条件。

if(m\_issleep==false&&distance\_to\_point(player.x, player.y)==0)

{ instance\_create\_depth(x, y,-1, obj\_enemy3\_boom\_area);

hp=0; }

if(hp==0)

{instance\_destroy();}

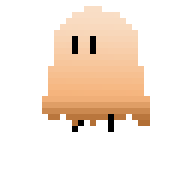
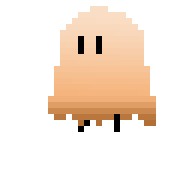
由于前面已经定义了enemy3的碰撞体积，所以这里我又创建一个大一点的碰撞盒子，然后在执行爆炸时调用它，同时enemy3死掉消失。在主角那里在添加与改盒子碰撞后减少的血量事件代码。这样enemy3就完成了，虽然做大比较简单，但是还是非常有成就感的。

第十周

既然完成了怪物，那就自然少不了让主角去攻击敌人的判定，在让三个组员设计怪物的同时，我也没有闲着完成了主角的攻击动画以及攻击所产生的剑气特效，准备完成之后我们就开始了攻击部分的代码编写。首先，我们给主角设定了攻击的按键为J，也就是当玩家按下J之后，主角就会攻击敌人，对应的代码如下：

else if keyboard\_check(ord("J"))

{



switch(m\_playerDirection)

{

case PlayerDirection.UP:

sprite\_index = spr\_hero\_attack\_up;;

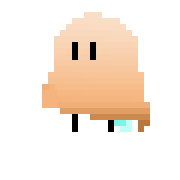
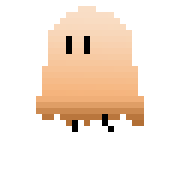
break;

case PlayerDirection.DOWN:

sprite\_index = spr\_hero\_attack\_down;;

break;

case PlayerDirection.LEFT:



sprite\_index = spr\_hero\_attack\_side;;

break;

case PlayerDirection.RIGHT:

sprite\_index = spr\_hero\_attack\_side;;

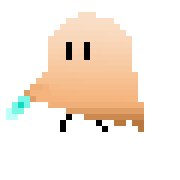
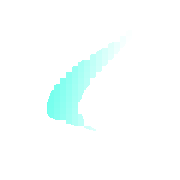
break;

}

这里我们引入了一个新的变量：m\_playerDirection，这个变量的完整定义如下：

enum PlayerDirection{

UP,



DOWN,

LEFT,

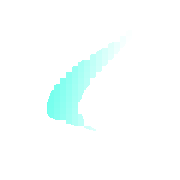
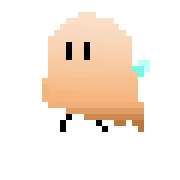
RIGHT

}

m\_playerDirection = PlayerDirection.DOWN; //默认向下

这个变量是用于判定主角面对的方向以便于在按下J时播放对应的攻击动画。但这个代码有一个问题，那就是如果我们按下J之后马上放开，攻击动画并不会完整的播放完。而我们需要的是一旦按下J，主角就会完成一整个攻击流程，所以这就需要我们在if语句中加一个判定，防止动画播放不完全，这里就引入另一个新的变量开关m\_isAttacking，这个变量的作用有两个，一个是防止动画中断，另一是使得攻击动画播放时玩家做出其他动作不会有响应。因此我们在switch之前加入这一段代码：m\_isAttacking = true;用于攻击时打开这个开关。并且在整个step中加一个最外层的if语句：

if(m\_isAttacking == false)



{

……

} //也就是当角色攻击时，玩家操作无响应

当然，当攻击完成之后，我们要关闭这个变量，这里我们引入了一个新的函数：animationend，这个函数在每个动画播放完之后都会进行一次响应，我们在这个函数下写入如下代码：

if(m\_isAttacking && sprite\_index == spr\_hero\_attack)

{

m\_isAttacking = false;

}

也就是控制这个变量开关的关闭。动画处理完成之后，我们还需要为剑气的产生编写代码，剑气对应的obj是obj\_bladeeffect。那么为了调用这个obj，我们需要写如下代码：

if(sprite\_index == spr\_ysera\_attack)

{

if(image\_index == 5) //在动画的第五帧产生剑气

{

instance\_create(x, y, obj\_bladeeffect);

}

if(image\_index == 8) //在动画的第八帧剑气消失

{

instance\_destroy(obj\_bladeeffect);

}

}

这样攻击的动画部分算是告一段落了。

最后一周

终于到了最后的一个礼拜了，这几个礼拜大家都在忙着期末的论文和复习，但大家依旧是忙里偷闲抽出时间来码最后的代码，这些代码也可以说是游戏中不可或缺的部分：攻击判定。现在我们的主角虽然有了攻击的动画和剑气，但是显然这些东西的判定我们都没有做，也就是说主角做出攻击动画之后怪物并不会有任何反应。反之，敌人的攻击打到主角的身上主角也不会有伤血之类的判定，这个是很不科学的。首先我们为主角和怪物添加血量，在他们的create语句中添加如下代码：

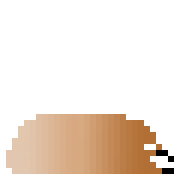
m\_herohp = 3; //设定血量

m\_isDead = false; //设定一个开关，当死亡后不再进行任何判定

之后我们为主角和怪物设定死亡的动画触发，在step语句中添加：

if(m\_isDead)

return 0;



if(m\_hp == 0){

m\_isDead = true;

sprite\_index = spr\_hero\_die;

return 0;

}

那么这样一来血量判定就做好了，但我们还需要做伤血的判定，这一部分并不难。首先我们给敌人以及主角攻击发出的剑气添加一个对应的hitbox，并创建一个变量m\_enemy用来使hitbox跟随敌人，并使之在与主角发生碰撞时产生判定，之后我们在hitboxondamage这个函数中添加如下代码：

if(instance\_exists(m\_enemy))

m\_herohp = m\_herohp - 1;;

}

这样一来，最简单的伤血判定就做好了。然而我们已经没有时间来制作可视化的血量以及更多更完善的动画和判定，可以说是真的真的十分遗憾。

写在最后的话

不得不说这段时间对游戏开发的摸索实在是让我受益匪浅，不仅让我对游戏开发有了最基本的了解，还让我熟悉了与之相关的不少代码和软件。要说唯一的遗憾的话那就是时间还是太紧了，毕竟对于学生的我们来说，每个礼拜大家能够聚在一起讨论游戏制作的时间真的十分有限，再加上我们都是初学者，磕磕绊绊一路走来效率实在说不上有多高，花了十几天也只能做出一个游戏最最基本的雏形。但即使是这样，我依旧对我们这几周的努力感到自豪，并且在这几周的游戏开发过程中，我们是真正地感受到了游戏开发的魅力，乐在其中。我想，在接下去的空余时间，我还会继续研究GMS这款软件，毕竟我们现在只是接触到了它的冰山一角，当然我也会乐于接触其他的游戏制作软件，开拓我的视野。是实训让四个懵懂的少年鼓起勇气踏出了追梦的第一步，想必这定会成为我们人生中不可磨灭的记忆。

特别鸣谢

Indienova独立游戏社区 以及其撰稿人 青铜的幻想

在我们陷入瓶颈时，是幻想大大的教程帮助了我们，真的十分感谢。