**Unity3D教程：实现攻击扣血的流程**

Posted on 2013年01月10日 by U3d / [Unity3D脚本/插件](http://www.unitymanual.com/category/script)/被围观 504 次

这个流程主要实现以下功能：界面上显示玩家当前的总血量，当被其他玩家发射的子弹打中后，会减1格血，血量为0时判定为死亡，会重置玩家的位置到出生点，并且重设血量为最大血量。

实现这个逻辑，分了以下几个步骤。

1.维护玩家血量相关信息。

首先玩家血量要用图形化在界面上显示，首先需要在场景中建立一个GUITexture，这里起名为GUI\_heart,来表示当前血量图片。为了控制血量的显示，为GUI\_heartf附加了一个控制脚本，名为HeartControl.cs。这个脚本的内容如下：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | **using** UnityEngine; |
| 02 | **using** System.Collections; |
| 03 | **public** **class** HeartControl : MonoBehaviour { |
| 04 | **public** Texture2D[] heartImageArray; |
| 05 | **private** **const** **int** maxLives = 5; |
| 06 | **private** **static** **int** lives = maxLives; |
| 07 | **public** **static** **void** DecreaseLive() |
| 08 | { |
| 09 | lives--; |
| 10 | **if**(lives==0) |
| 11 | { |
| 12 | lives=maxLives; |
| 13 | PlayerManager.selfPlayer.position = GameObject.Find("SpwanPlayer").transform.position; |
| 14 | } |
| 15 | } |
| 16 | *// Use this for initialization* |
| 17 | **void** Start () { |
| 18 | } |
| 19 | *// Update is called once per frame* |
| 20 | **void** Update () { |
| 21 | **if**(PlayerManager.selfPlayer!=**null**) |
| 22 | { |
| 23 | guiTexture.enabled = **true**; |
| 24 | guiTexture.texture = heartImageArray[lives-1]; |
| 25 | } |
| 26 | **else** |
| 27 | { |
| 28 | guiTexture.enabled = **false**; |
| 29 | } |
| 30 | } |
| 31 | } |

这里heartImageArray表示不同血量所对应的血量图片，血量用心型图标来显示，5格血就是5颗心。把这个数组声明为public,可以在编辑器中，设置这个数组的尺寸，以及每个入口点所对应的图片，十分方便。

maxLives就是最大血量有5格血，lives是当前人物的血量。

在每帧更新函数Update()中，首先检查当前玩家的变量是否为空，为空可能是因为是Server端，这种情况下就不需要显示血量，因此将图片设为false：guiTexture.enabled = false;

如果不为空，那么将根据当前血量的值，把对应数组中的图片赋值给这个guiTexture当前的texture,这样就实现图片随血量值的变化而变化。

这里PlayerManager.selfPlayer中的PlayerManager我自己建立的一个全局的静态类，维护了唯一的静态变量private static Transform \_selfplayer，代表客户端所对应的玩家自己。这个类可以被所有脚本所访问，访问起来十分方便，实现如下：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | **using** UnityEngine; |
| 02 | **using** System.Collections; |
| 03 | **using** System; |
| 04 | **public** **static** **class** PlayerManager { |
| 05 | **private** **static** Transform \_selfplayer = **null**; |
| 06 | **public** **static** Transform selfPlayer |
| 07 | { |
| 08 | **get** |
| 09 | { |
| 10 | **return** \_selfplayer; |
| 11 | } |
| 12 | **set** |
| 13 | { |
| 14 | \_selfplayer = **value**; |
| 15 | } |
| 16 | } |
| 17 | } |

后期随着逻辑的复杂，可以为这个脚本的玩家增加更多的数据来维护。

然后是DecreaseLive函数，这个函数可以被外部所调用，给玩家减血，调用后，自动扣除一滴血,为0时，正如规定的一样，重设玩家血量为最大血量，并且找到重生点的位置，将玩家位置重置。

2.控制子弹打中玩家的逻辑

实现大体流程是先给玩家的人物身上附加一个释放子弹的脚本，当客户端自己的玩家按键触发技能时，就调用一个网络上的RPC，告诉所有客户端和服务器在本地创建一个飞行的子弹。并且在这个人物脚本里添加一个响应人物碰撞的事件函数，然后检查是不是碰撞的对象是子弹，而且不是这个人物所发射的，如果检测成功，就调用HeartControl脚本的减血函数。

这个脚本名为Shoot.cs,实现如下：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | **using** UnityEngine; |
| 02 | **using** System.Collections; |
| 03 | **public** **class** Shoot : MonoBehaviour |
| 04 | { |
| 05 | **public** Texture skillTex1; |
| 06 | **public** Texture skillTex2; |
| 07 | **public** Transform bulletPrefab; |
| 08 | **public** Transform explosionEffect; |
| 09 | **public** AnimationClip attackAnim; |
| 10 | *// Use this for initialization* |
| 11 | **void** Start () |
| 12 | { |
| 13 | } |
| 14 | *// Update is called once per frame* |
| 15 | **void** Update () |
| 16 | { |
| 17 | **if** (networkView.isMine && Input.GetKeyUp ("2")) { |
| 18 | ShootBullet (); |
| 19 | } |
| 20 | **if** (networkView.isMine && Input.GetKeyUp ("3")) { |
| 21 | ExplosiveEffect(); |
| 22 |  |
| 23 | } |
| 24 | } |
| 25 | **void** OnGUI () |
| 26 | { |
| 27 | **if** (networkView.isMine) |
| 28 | { |
| 29 | **if**(GUI.Button (new Rect ((**float**)(0.5\*Screen.width&minus;60), (**float**)(Screen.height&minus;60), 60, 60), skillTex1)) |
| 30 | { |
| 31 | ShootBullet();} |
| 32 | **if**(GUI.Button(new Rect((**float**)0.5\*Screen.width,(**float**)Screen.height&minus;60,60,60),skillTex2)) |
| 33 | { |
| 34 | ExplosiveEffect(); |
| 35 | } |
| 36 | } |
| 37 | } |
| 38 | **void** OnCollisionEnter(Collision collisionInfo) |
| 39 | { |
| 40 | **if**(collisionInfo.gameObject.tag=="bullet" ) |
| 41 | { |
| 42 | BulletScript bs = (BulletScript)collisionInfo.gameObject.GetComponent("BulletScript"); |
| 43 | **if**(bs.Owner != gameObject) |
| 44 | { |
| 45 | print("OnCollisionEnter"+collisionInfo.gameObject.name); |
| 46 | Destroy(collisionInfo.gameObject); |
| 47 | **if**(!networkView.isMine) |
| 48 | **return**; |
| 49 | HeartControl.DecreaseLive(); |
| 50 | } |
| 51 | } |
| 52 | } |
| 53 | **void** ExplosiveEffect() |
| 54 | { |
| 55 |  |
| 56 | networkView.RPC("SpwanExplosion",RPCMode.All); |
| 57 | SendMessage("PlayAnimation\_Attack"); |
| 58 | } |
| 59 | **void** ShootBullet() |
| 60 | { |
| 61 | networkView.RPC("SpawnBullet",RPCMode.All); |
| 62 | } |
| 63 | [RPC] |
| 64 | **void** SpawnBullet () |
| 65 | { |
| 66 |  |
| 67 | Vector3 forward = transform.TransformDirection (Vector3.forward); |
| 68 | Transform effectPoint = transform.Find ("effectPoint"); |
| 69 | PrefabDepends preScript = (PrefabDepends)GameObject.Find("GameObject\_GlobalController").GetComponent("PrefabDepends"); |
| 70 | **if** (effectPoint) { |
| 71 |  |
| 72 | Transform bullet = (Transform)Instantiate (preScript.bulletPrefab, effectPoint.position, Quaternion.identity); |
| 73 | bullet.rigidbody.AddForce (forward \* 2000); |
| 74 | BulletScript bs = (BulletScript)bullet.GetComponent("BulletScript"); |
| 75 | bs.Owner = gameObject; |
| 76 | } |
| 77 |  |
| 78 | } |
| 79 | [RPC] |
| 80 | **void** SpwanExplosion() |
| 81 | { |
| 82 | Vector3 forward = transform.TransformDirection (Vector3.forward); |
| 83 | Transform effectPoint = transform.Find ("effectPoint2"); |
| 84 | **if** (effectPoint) { |
| 85 | *//Object temp = Network.Instantiate (bulletPrefab, effectPoint.position, Quaternion.identity, 0);* |
| 86 | Transform explosion = (Transform)Instantiate (explosionEffect, effectPoint.position, Quaternion.identity); |
| 87 | *//Network.Instantiate(bulletPrefab,transform.Find("effectPoint").position,Quaternion.identity,0);* |
| 88 | } |
| 89 | } |
| 90 | } |
| 91 |  |

SpawnBullet就是调用的RPC call,在这里除了用子弹的prefab创建出子弹附加初始速度，还给子弹的脚本BulletScript的一个变量Owner赋值为脚本所附加到的玩家，这是为了判断这个子弹是哪个玩家发出用的。

碰撞检测的函数是OnCollisionEnter，要想触发这个函数，人物身上需要有一个Collider控件，而CharacterController是没有用的。所有的子弹都设为了名为"bullet"的tag,所以在这里检测，如果碰撞的对象是bullet,并且子弹上的脚本的变量Owner不是这个玩家自己，说明被别人的子弹击中，就先销毁本地创建出来的子弹，然后判断如果这个玩家是这个客户端所对应的，就进行减血。这个判断是必要的，因为这里的血量是全局静态的，只有一份代表自己，其他玩家被子弹攻击到了，不需要更新自己的血量。