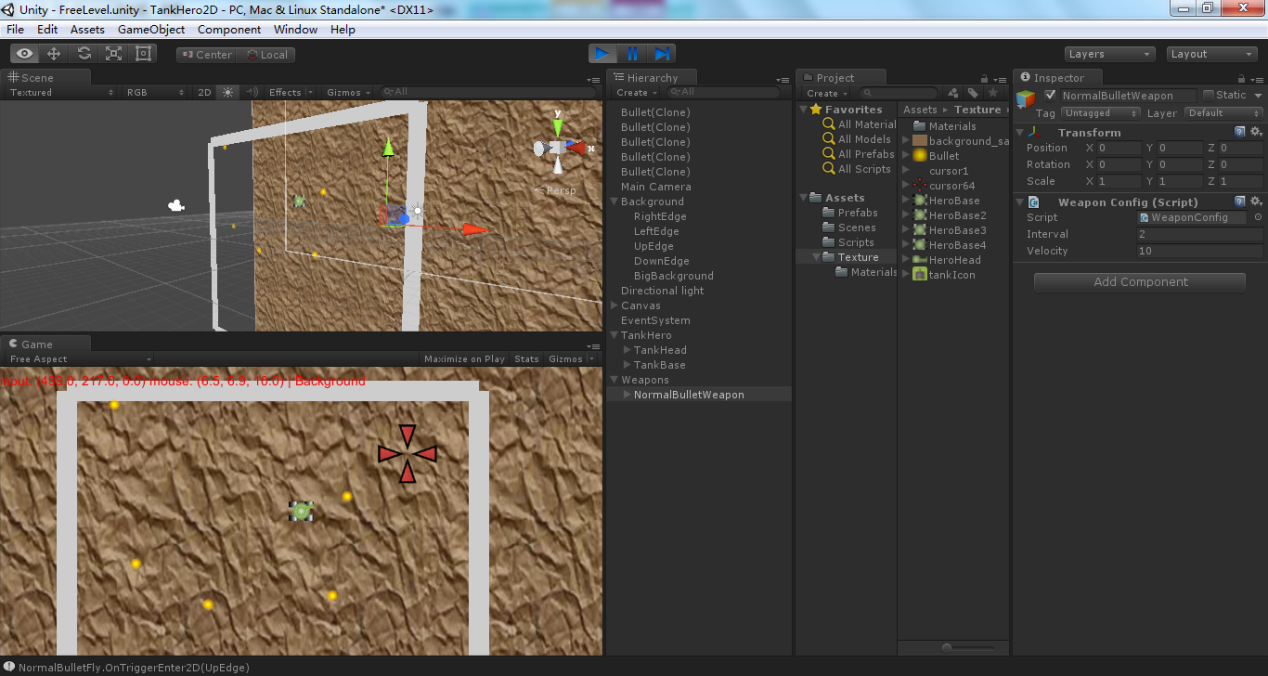
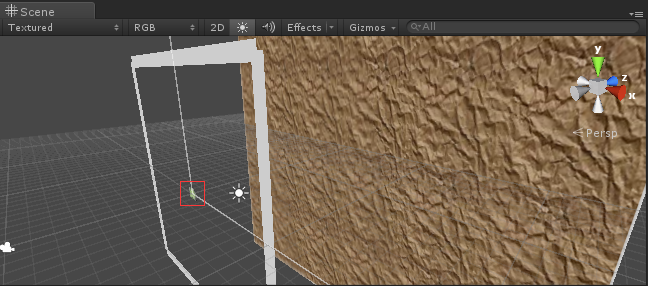
自制Unity小游戏TankHero-2D(1)制作主角坦克

我在做这样一个坦克游戏，是仿照（<http://game.kid.qq.com/a/20140221/028931.htm>）这个游戏制作的。仅为学习Unity之用。图片大部分是自己画的，少数是从网上搜来的。您可以到我的github页面（<https://github.com/bitzhuwei/TankHero-2D>）上得到工程源码。

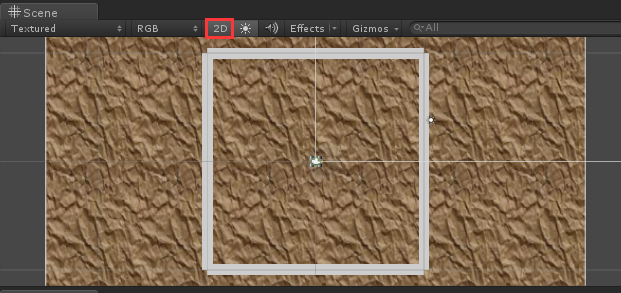


本篇主要记录制作主角坦克（TankHero）的一些重点。

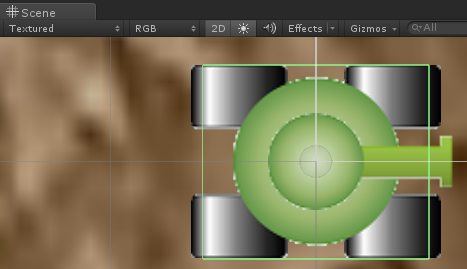
# 2D游戏布局



如上图所示，红色矩形围起来的是主角坦克，白色的一圈是围墙，坦克和围墙在同一平面上。地面背景放到离摄像机最远的后方。这样，在2D摄像机下看起来是这样的：

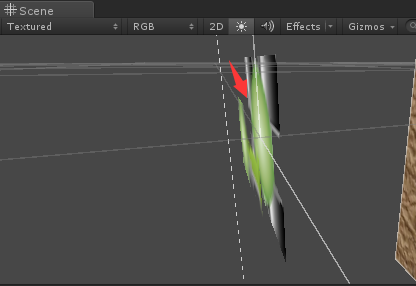


坦克本身由底座（Base）和炮塔（Head）两部分组成。当然，在2D世界，其实就是两个扁平的贴图。在2D摄像机下是这样的：



（PS：上图中的绿色矩形框是Box Collider 2D，忽略即可）

为了保证炮塔始终显示在底座上方，我们要让炮塔稍微靠近一点摄像机。如下图所示，炮塔和底座两张贴图是分隔开的。



# 坦克的运动

坦克的运动包括：上下左右平移；底座旋转；炮塔旋转。其中平移时会同样地移动底座和炮塔，所以用最上层的TankHero负责。底座和炮塔的旋转我们要求两者互不干涉，所以TankHead和TankBase放在同一层，并且分别负责各自的旋转。



## 移动

坦克的移动十分容易。玩家在纵横方向的按键情况就是坦克的移动方向，速度由程序员指定，再乘上时间就好了。

|  |
| --- |
| void Update () {  var h = Input.GetAxis ("Horizontal");  var v = Input.GetAxis ("Vertical");  if (Mathf.Abs(h) > Quaternion.kEpsilon || Mathf.Abs(v) > Quaternion.kEpsilon)  {  Move (h, v);  }  }  void Move(float h, float v)  {  var moveVector = new Vector3 (h, v, 0);  moveVector.Normalize ();  this.transform.position += moveVector \* speed \* Time.deltaTime;  } |

## 底座旋转

底座应该朝向移动的方向，即上文的moveVector。这里用Quaternion.Slerp使底座平滑地转向moveVector。

|  |
| --- |
| void Update () {  var h = Input.GetAxis ("Horizontal");  var v = Input.GetAxis ("Vertical");    if (Mathf.Abs(h) > Quaternion.kEpsilon || Mathf.Abs(v) > Quaternion.kEpsilon)  {  this.targetAngle = Mathf.Atan2(v, h) \* Mathf.Rad2Deg;  //Debug.Log("target angle: " + targetAngle);  }    this.transform.rotation = Quaternion.Slerp (  this.transform.rotation,  Quaternion.Euler (0, 0, targetAngle),  rotationSpeed \* Time.deltaTime);  } |

## 炮塔旋转

炮塔要指向鼠标（即目标）所在的位置，所以从炮塔到鼠标的向量就是炮塔的方向。

注意炮塔不是围绕自身的中心旋转的，这个旋转点需要根据坦克的形状来指定。所以这里要用**transform.RotateAround**来进行旋转。

|  |
| --- |
| void Update () {  Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);  RaycastHit hit;  if(Physics.Raycast(ray, out hit))  {  var p = hit.point;  var y = p.y - this.transform.position.y;  var x = p.x - this.transform.position.x;  if (Mathf.Abs(y) > Quaternion.kEpsilon || Mathf.Abs(x) > Quaternion.kEpsilon)  {  this.targetAngle = Mathf.Atan2(y, x) \* Mathf.Rad2Deg;  var angle = this.targetAngle - this.transform.rotation.eulerAngles.z;  this.transform.RotateAround (this.rotationCenter.position, new Vector3 (0, 0, 1), angle);  }  }  } |

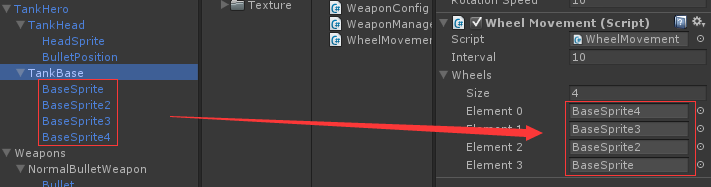
## 车轮滚动

其实这不算是运动了，不过放在这一节也还算紧凑。

坦克移动的时候，我希望车轮下下图所示这样，显得很生动：

|  |
| --- |
| C:\Users\DELL\Pictures\TankHero\TankMove.gif |

我的思路是用4张图片表现车轮滚动的效果，让TankBase负责循环显示这4张图片。

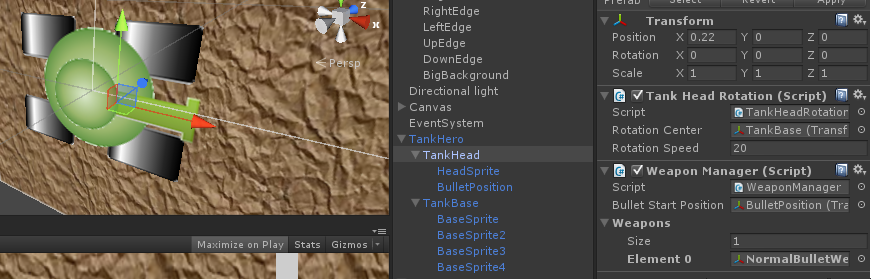


当然，脚本可以处理任意多张图片的循环播放。其关键就是依次将各个BaseSprite的**renderer.enabled**字段设为true。

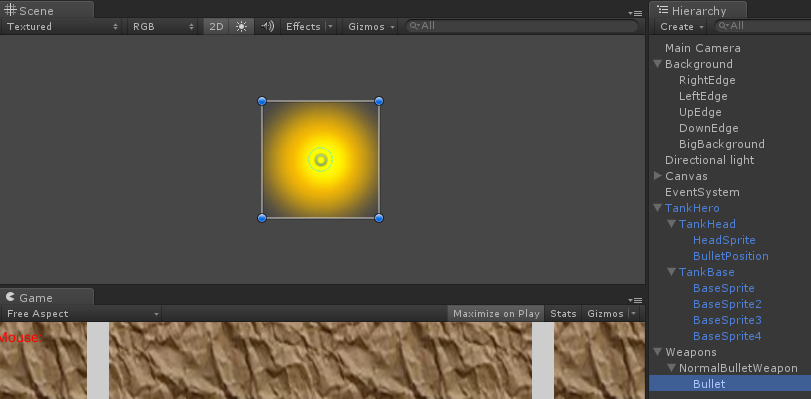
|  |
| --- |
| public float interval = 10;  public List<GameObject> wheels;  private int current = 0;  private float passedInterval = 0;  // Use this for initialization  void Start () {  if (wheels != null && wheels.Count > 0)  { wheels[0].renderer.enabled = true; }  for (int i = 1; i < wheels.Count; i++)  {  wheels[i].renderer.enabled = false;  }  }    // Update is called once per frame  void Update () {  if (wheels == null || wheels.Count < 2) { return; }  var h = Input.GetAxis ("Horizontal");  var v = Input.GetAxis ("Vertical");    if (Mathf.Abs(h) > Quaternion.kEpsilon || Mathf.Abs(v) > Quaternion.kEpsilon)  {  passedInterval += Time.deltaTime \* 100;  //Debug.Log (passedInterval);  if (passedInterval >= interval)  {  var tmp = current;  if (current == wheels.Count - 1) { current = 0; }  else { current++; }  wheels[current].renderer.enabled = true;  wheels[tmp].renderer.enabled = false;  passedInterval = 0;  }  }  } |

# 坦克开炮

这个游戏中，TankHero能够发射多种炮弹，所以需要有多种武器，每种武器发射一种炮弹。因此炮塔充当了武器管理员的角色，而不是武器本身。一种武器决定了它发射的炮弹的速度、威力等信息。

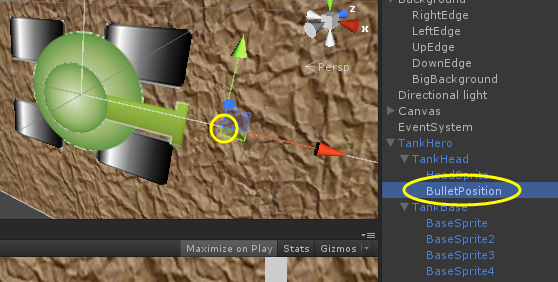


发射炮弹这种事，典型的方法是用**Instantiate**方法。这就需要在场景中持有一个现成的炮弹。如下图所示：



这个炮弹要永远存在，还不能被摄像机看到，所以我们把它放到之前说的地面背景的更后面。

你注意到图中的炮弹中心有个比较小的绿色的圈，这个圈是Circle Collider 2D，是用来产生碰撞的。我刻意把这个Collider调到这么小，是为了避免在坦克刚刚发射出炮弹时，炮弹与自身产生碰撞（即自己开炮瞬间打了自己）。



同时，在上图中我用黄色圈圈出了那个BulletPosition的gameobject，这是专门用来指定炮弹产生点的，也是为了避免炮弹刚刚发射出来就把自己给打了。

注意，带2D的Collider似乎有这样的问题：无论在Z方向上是否在同一平面，都能引发碰撞事件。所以，那个永生的炮弹，虽然藏到地面背景后方去了，却仍旧可能与游戏中的其它物体发生碰撞（然后就会爆炸消失被Destroy掉，之后就无法再用Instantiate来创建炮弹了）。为了避免它的Destroy，我们需要将它和其它炮弹区别开来，所以就必须给炮弹对象添加一个**undying**字段，让**undying**为true的炮弹在触发了碰撞事件时也不爆炸消失。

# 摄像机随主角移动

我希望地图能够大一点，所以一屏肯定放不下。所以需要摄像机随主角坦克的移动而移动。这个很容易，不断跟随主角坦克就行了。

|  |
| --- |
| public float catchingSpeed = 1;  private Transform tankHero;  void Awake()  {  this.tankHero = GameObject.FindGameObjectWithTag (Tags.hero).transform;  }  void Update () {  var targetPosition = new Vector3 (this.tankHero.position.x, this.tankHero.position.y, this.transform.position.z);  this.transform.position = Vector3.Lerp (this.transform.position, targetPosition, Time.deltaTime \* this.catchingSpeed);  } |

注意这里将**catchingSpeed**调低一些，会产生摄像机延迟跟随主角坦克的现象。我很喜欢这种跟随的感觉，柔和不生硬，而且还解决了后文遇到的一个问题。

# 自定义鼠标样式

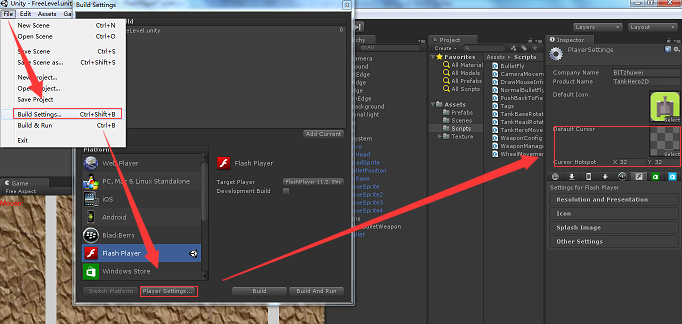
我希望鼠标在游戏中显示为下图所示的样子，很带感。



方法有两种。

## Default cursor

一是在File – Build Settings – Player Settings打开的Inspector面板中设置Default Cursor。



这个方法有点问题，首先在build之后的exe中你可能发现鼠标彻底消失了，既没有原始图标也没有自定义图标，其次在你修改了自定义图标之后，可能会显示成一个很奇怪的图标，最后，这样自定义的图标，其清晰度大打折扣，其size也是固定的。

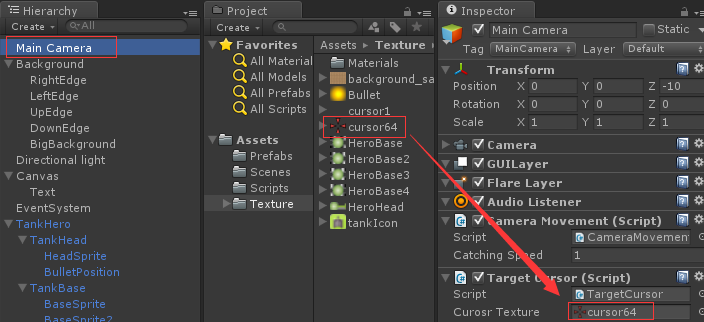
所以我推荐另一种方法，即用脚本实现。

## 脚本实现

典型的实现方式是这样的，在主摄像机上添加一个TargetCusor.cs的脚本（脚本名无所谓），编写代码如下：

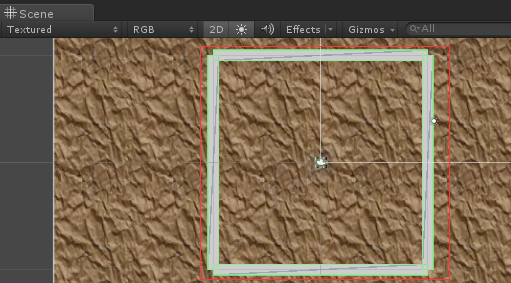
|  |
| --- |
| //3D贴图是Material,2D贴图是Texture  public Texture CurosrTexture;  void OnGUI() { // 渲染GUI和处理GUI时调用。  if (CurosrTexture != null) {  // 计算图片左上角的坐标  float left = Input.mousePosition.x - CurosrTexture.width / 2;  float top = Screen.height - Input.mousePosition.y - CurosrTexture.height / 2;    GUI.DrawTexture(new Rect(left, top, CurosrTexture.width, CurosrTexture.height), CurosrTexture);  }  } |

在Inspector面板中指定你的图标即可。



# 围墙

限制坦克和炮弹的活动范围是必须的。这里我暂且简单地制作一个正方形围墙。



这个围墙由四个quad组成。绿色的线条是Box Collider 2D组件。围墙的功能就是把撞上它的东西（坦克、炮弹等）弹回去。这里不得不用一个Dictionary<Collider2D, Vector3>字典记录撞到围墙的物体在碰撞瞬间的位置，因为之后要将物体弹回这个位置。

|  |
| --- |
| public class PushBackToField : MonoBehaviour {  Dictionary<Collider2D, Vector3> initialPositionDict;// = new Dictionary<Collider, Vector3>();  void Awake()  {  initialPositionDict = new Dictionary<Collider2D, Vector3> ();  }  void OnTriggerEnter2D(Collider2D other)  {  if (initialPositionDict.ContainsKey(other))  {  initialPositionDict[other] = other.transform.position;  }  else  {  initialPositionDict.Add(other, other.transform.position);  }  }    void OnTriggerStay2D(Collider2D other)  {  Push (other);  }    void OnTriggerExit2D(Collider2D other)  {  if (initialPositionDict.ContainsKey(other))  {  initialPositionDict.Remove(other);  }  }    void Push(Collider2D other)  {  Vector3 initialPosition = Vector3.zero;  if (initialPositionDict.ContainsKey(other))  {  initialPosition = initialPositionDict[other];  }  else  {  Debug.LogError(string.Format("{0} should have been added to the dict.", other.gameObject.name));  }    if ((initialPosition - other.transform.position).magnitude > 0.001f)  {  //Debug.Log("lerp push");  other.transform.position = Vector3.Lerp(other.transform.position, Vector3.zero, Time.deltaTime);  }  else  {  //Debug.Log("sudden push");  other.transform.position = initialPosition;  }  }  } |

这个脚本对上下左右四个围墙都适用，以后有了别的形状的围墙，也仍然适用。这也是它的优点之一。

说到这个围墙的反弹，就涉及摄像机跟随的一个问题。实际上，围墙反弹时，如果玩家持续撞击围墙，会使玩家坦克产生快速的震动。此时摄像机也就跟着快速震动，这很影响体验。上文里将跟随速度设置得比较低时，这种震动就不会影响到摄像机。这是因为，摄像机反应慢，震动速度快，不等摄像机需要向左跟随，就又要向右跟随了，所以摄像机基本上就在原地不动了。

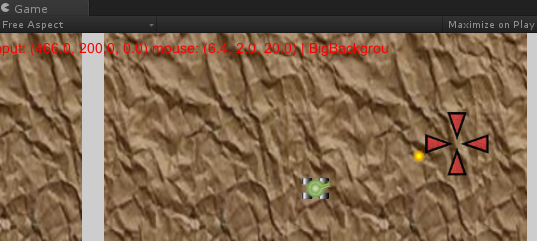
将上文的**catchingSpeed**调大一些，再持续去撞墙，你就会明白了。

# 光源

忘了说，要添加一个线光源，不然场景会很暗淡。下图就是没有添加光源的样子。

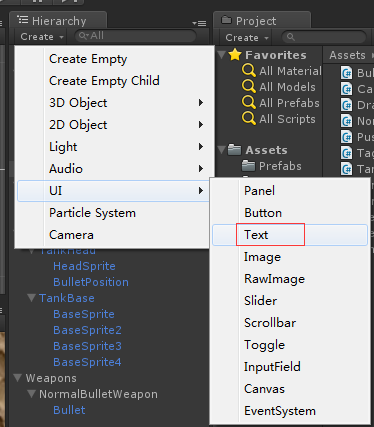


加上光源就成了这样：

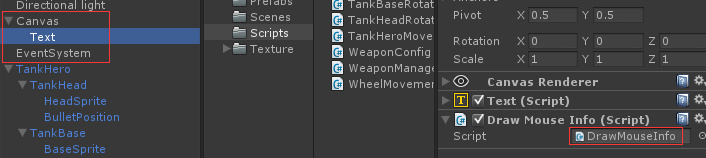


# 显示文字

想显示上图所示的文字？用Unity最近推出的uGUI还是很舒服的（也可能是因为我没有学过nGUI等等的UI系统吧）。



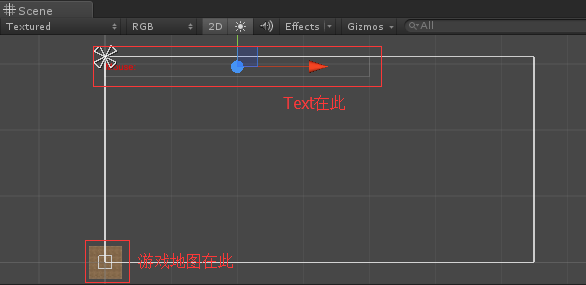
点击Text后，会增加3个对象，Canvas，Text和EventSystem。



给Text对象添加一个**DrawMouseInfo.cs**的组件（名字无所谓）。

|  |
| --- |
| public class DrawMouseInfo : MonoBehaviour {  Text guiText;  void Awake()  {  guiText = this.GetComponent<Text> ();  }  void Update () {  Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);  RaycastHit hit;  if(Physics.Raycast(ray, out hit))  {  guiText.text = string.Format ("input: {0} mouse: {1} | {2}", Input.mousePosition, hit.point, hit.transform.gameObject.name);  }  else  {  guiText.text = string.Format ("input: {0} mouse: {1} | {2}", Input.mousePosition, "null", "null");  }  }  } |

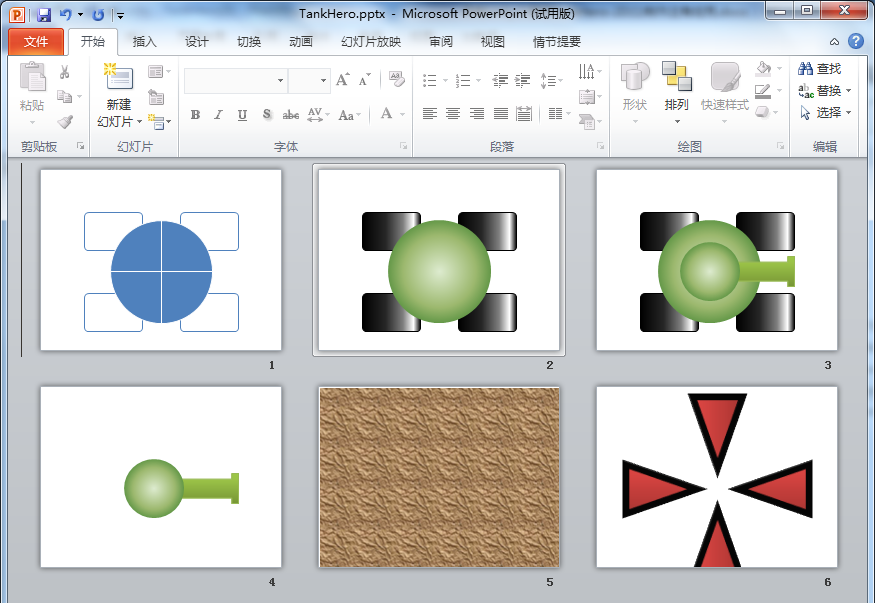
uGUI对象是可以在Scene视图里拖动的，只不过你要先找到它。



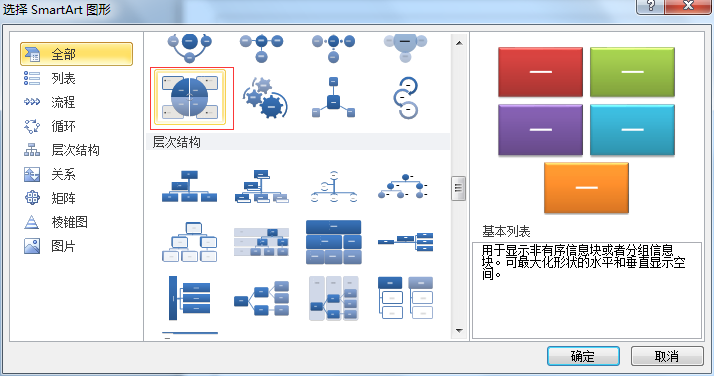
它的位置很奇葩，如上图所示，整个地图在它的Canvas脚下都很渺小。

# 快速自制贴图资源

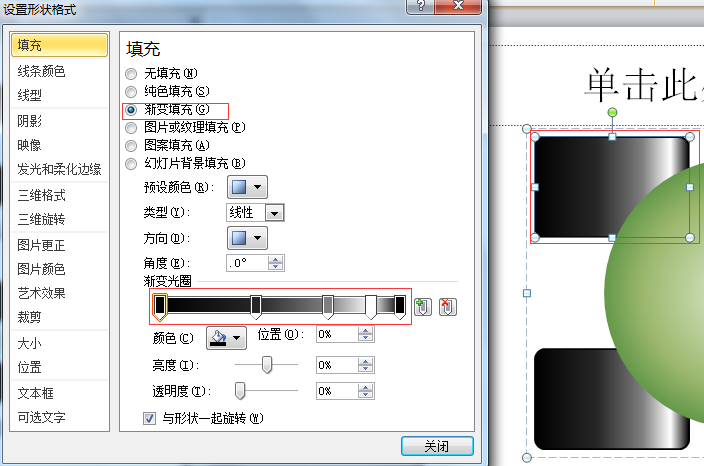
本项目中的坦克、子弹、光标、背景图都是本人制作的，制作工具你猜猜？是PPT。



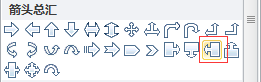
坦克底座是SmartArt图形里的。



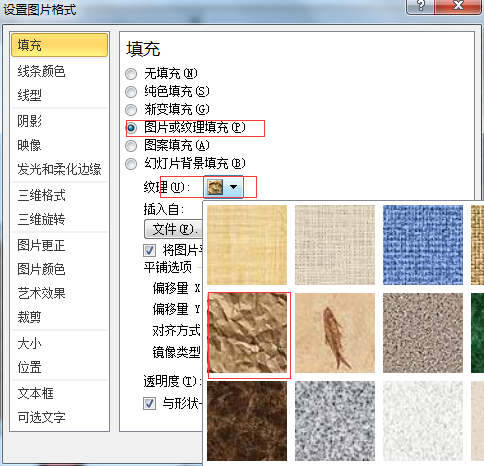
轮子只是设置了一下渐变填充。



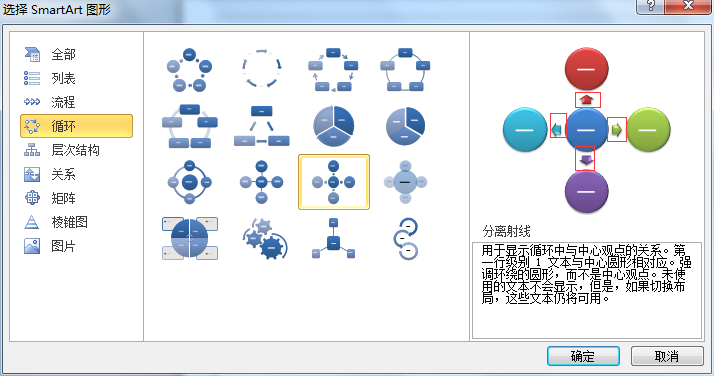
炮塔的圆形，把底座的圆形缩小一点就是。炮塔的炮管，是“形状”里的箭头，删掉凸起的尖的部分，调整一下锚点长短就OK。

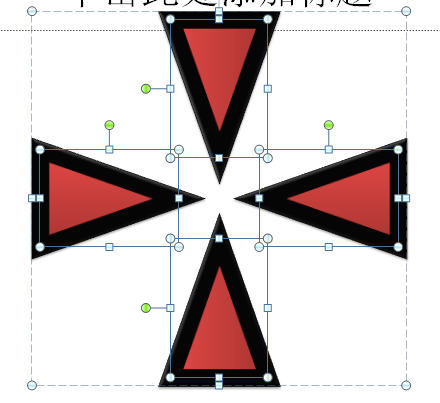


背景用的是“纹理填充”，看到第二行第一个了没？



准星，用的是SmartArt里的“分离射线”。把四个箭头留下，其它内容删除。再把箭头的尾部顶点删除，左右交换位置，上下交换位置，上个色就成了。





还可以吧？

# 总结

您可以到我的github页面（<https://github.com/bitzhuwei/TankHero-2D>）上得到工程源码。

请多多指教~