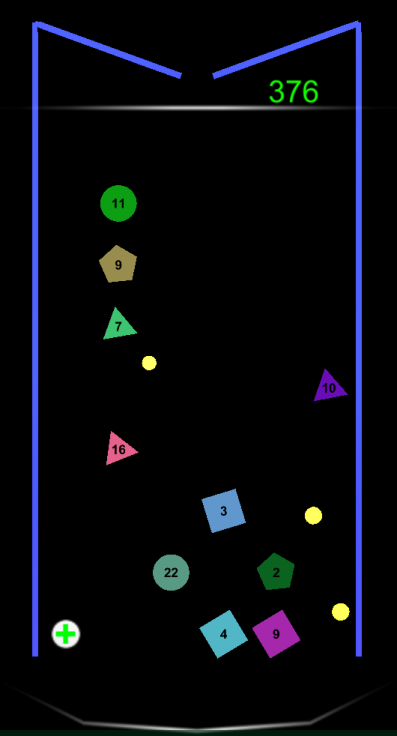
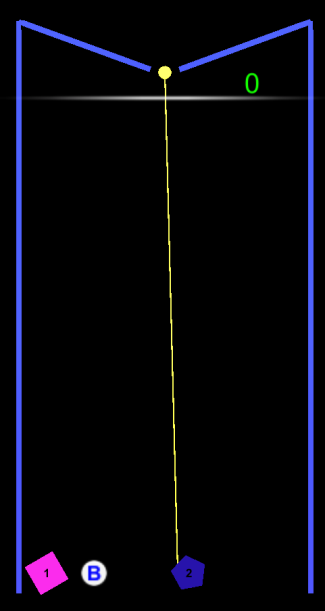
# 示例教程二—2D物理弹球

****

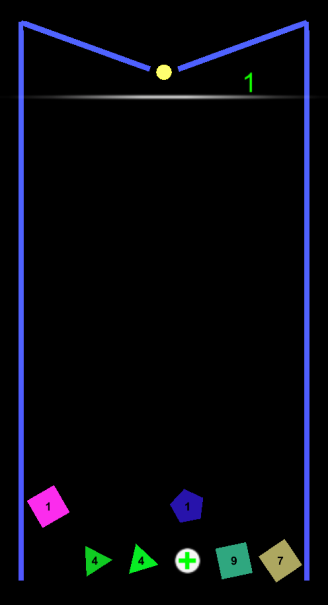
是否觉得很眼熟，这个例子是仿照流行的微信小游戏《弹一弹》和《弹球王者》，并用Unity引擎重新实现的。

## 游戏玩法

1. 最底层关卡随机生成几何图形。
2. 玩家从枪口发射小球进行打击消灭，几何图形上的数字代表血量，被打击一次数字-1，分数+1，数字为0时，该几何图形被消灭。

****

1. 每发射一次小球一回合结束，关卡网上走一层，底层再创建一次几何图形。

****

1. 在游戏中还有道具可以吃：

---变大道具：让该小球变大攻击力\*2，每个小球只能变大一次

****

---添加小球道具：在场景中增加一个小球供玩家使用

****

1. 当有几何图形到达顶层，则游戏结束，统计最终分数。

****

## 需求分析

那么这一款游戏有什么功能上的需求呢

1. 发射弹球——从枪口往鼠标所在方向发射
2. 关卡创建——随机创建敌人和道具
3. 场景交互——小球碰到不同类型物体会有不同效果
4. 关卡上升——每回合几何体和道具往上走一层
5. 状态判断——当前游戏是否结束
6. 得分显示——结算游戏分数

## 场景搭建

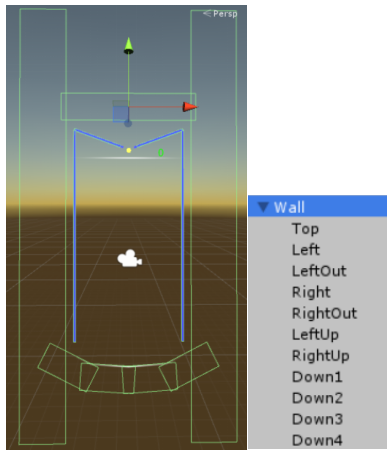
游戏属于2D游戏，场景我们都用2D精灵（Sprite）来搭建

### 砌墙（限定小球活动区域）

1. 我们创建一些空物体，挂上2D碰撞器

****

1. 给它们设置好位置和大小,在创建一个空的父物体Wall来管理它们



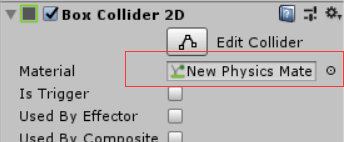
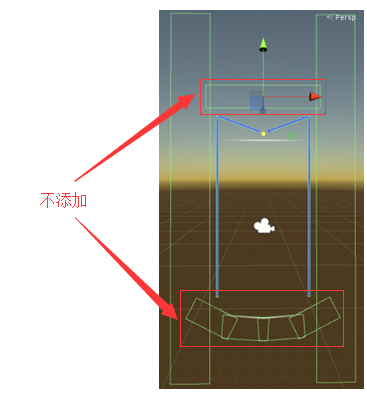
1. 创建2D物理材质，添加进2D碰撞器中可以使碰撞器拥有弹性

****

1. 点击它可以设置弹性

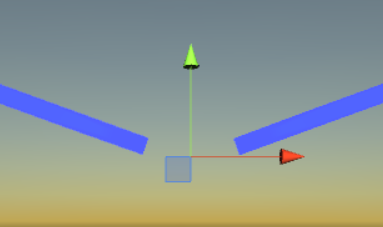
****

1. 除了顶上和底下的墙之外，其它墙的碰撞器里都添加上弹性材质

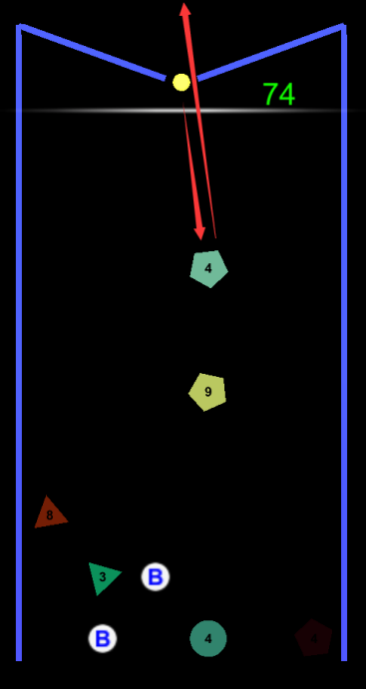
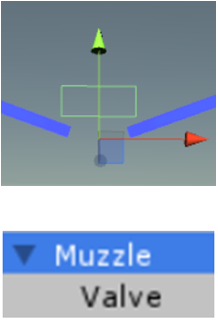
****

### 创建枪口（用于初始化小球发射位置）

1. 创建一个空物体作为枪口，取名“Muzzle”，放在指定位置

****

1. 为Muzzle做一个阀门取名“Valve”，作用是不让小球从枪口位置弹出去

** **

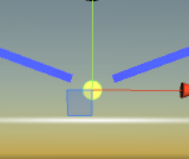
### 分数显示

创建一个空物体Score，再创建一个用来显示分数的Text，取名为“CurrentScore”

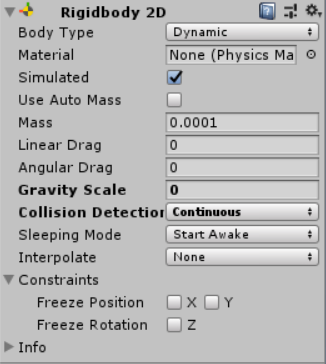
****

### 创建小球

1. 使用2D精灵创建一个小球，取名为“Ball”

****

1. 为小球添加刚体组件，做好如下设置

****

1. 创建一个空物体Balls来管理小球

****

1. 小球的脚本我们过一会儿再来完成

### 创建道具预制件

我们目前的道具有两款，一款变大道具和增加小球道具

1. 我们先来做变大道具BigProp，首先做一个精灵，上面的字母用Text做就行了

****

1. 这个道具的作用是小球碰到它后变大，攻击力翻倍，需要创建一个脚本挂它身上，这个脚本可以这样写：

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /// <summary>  ///挂变大道具上  /// </summary>  public class BigBall : MonoBehaviour  {  private void OnCollisionEnter2D(Collision2D collision) //被碰撞时调用  {  if (collision.gameObject.tag != "BigBall") //当普通小球碰到时  {  collision.transform.localScale \*= 1.2f;//小球变大20%成大球  collision.gameObject.tag = "BigBall"; //大球的标签设为“BigBall”  }  Destroy(gameObject); //销毁变大道具  }  } |

1. 再来做一个添加小球道具，该道具如果被小球碰到，则增加一个小球供玩家使用（当大球碰到它时也只增加一个小球）

****

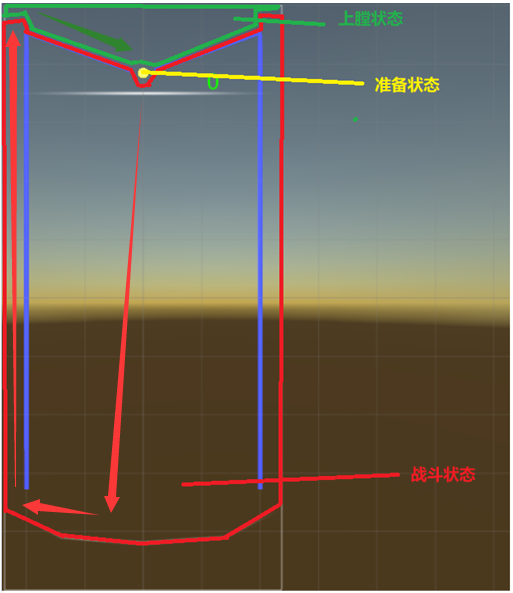
1. 添加小球道具也要挂一个脚本，脚本内容如下：

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /// <summary>  /// 挂复制道具上  /// </summary>  public class CopyBall : MonoBehaviour  {  //小球需要做一个预制件拖进来  public Transform ball;  private void OnCollisionEnter2D(Collision2D collision)//被小球碰到时调用  {  //获取小球transform组件  Transform tf = collision.transform;  //在小球位置复制一个新小球(小球预制件)  Transform newBall = Instantiate(ball, tf.position,tf.rotation);  //新小球认小球的父物体“Balls”为自己的父物体  newBall.parent = tf.parent;  //新小球往右弹开  newBall.GetComponent<Rigidbody2D>().AddForce(transform.right \* 0.02f);  //旧小球往左弹开  tf.GetComponent<Rigidbody2D>().AddForce(-transform.right \* 0.02f);  //销毁该道具  Destroy(gameObject);  }  } |

1. 把变大道具和添加道具都做成预制件

****

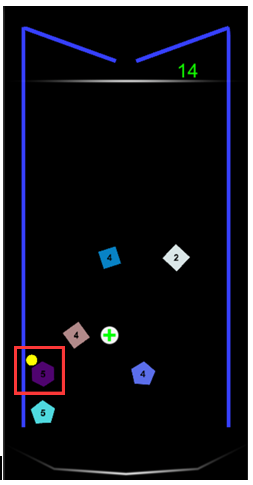
1. 我们来继续完成小球的功能，首先为小球创建一个小球状态机，用于记录和存储小球当前状态，当小球处于不同的状态时可以拥有不同的属性的，根据小球的当前位置改变状态：



1. 也就是说，小球在枪口位置是处于准备状态，此时的小球可以被发射，当进入红色战斗区域时，处于战斗状态，会按照红色箭头寻路回到顶上，到达顶层时处于上膛状态，会忘枪口处移动，创建状态机脚本

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /// <summary>  /// 小球状态机,不用挂在任何物体上  /// </summary>  public enum BallState  {  Ready, //准备阶段  Battle, //战斗阶段  Bore, //上膛阶段  } |

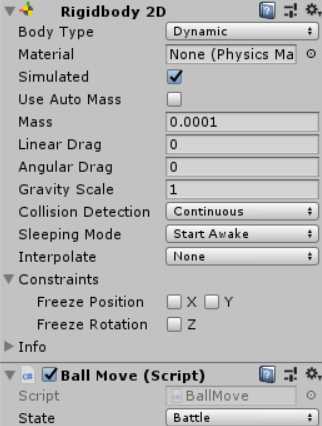
1. 如果小球状态机这里不是很懂，可以暂时忽略，这需要结合到后面的代码来理解
2. 创建小球的脚本时，有一些注意事项，小球有可能会被卡住不动：

****

我们创建小球脚本时要考虑卡住的解决办法，脚本详情如下：

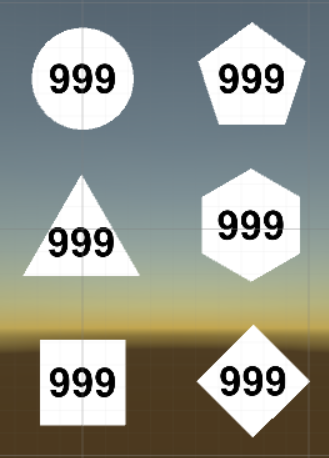
|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  /// <summary>  /// 挂小球上  /// </summary>  public class BallMove : MonoBehaviour  {  float timer; //计时用，记录小球卡住时间  public BallState state = BallState.Ready; //初始状态为准备状态  //碰撞时调一次，用于打击几何体（敌人）  private void OnCollisionEnter2D(Collision2D collision)  {  if (state == BallState.Battle) //如果在战斗阶段  {  GetComponent<Rigidbody2D>().gravityScale = 1; //碰到东西后重力为1  if (collision.gameObject.tag == "Enemy") //如果碰到敌人  {  //获取敌人数字  Text enemyNumber = collision.transform.GetChild(0).GetComponent<Text>();  //获取当前分数  Text Score = GameObject.Find("ScoreText").GetComponent<Text>();  if (tag == "BigBall") //如果自己是大球  {  //敌人数字-2  enemyNumber.text = ((System.Convert.ToInt32(enemyNumber.text)) - 2).ToString();  //当前分数+2  Score.text = ((System.Convert.ToInt32(Score.text)) + 2).ToString();  }  else //如果自己是小球  {  //敌人数字-1  enemyNumber.text = ((System.Convert.ToInt32(enemyNumber.text)) - 1).ToString();  //当前分数+1  Score.text = ((System.Convert.ToInt32(Score.text)) + 1).ToString();  }  }  }  }  //碰撞时持续调用,防止小球被卡住  private void OnCollisionStay2D(Collision2D collision)  {  if (collision.gameObject.tag == "Enemy") //如果和敌人持续碰撞  {  timer += Time.deltaTime; //开始计时  if (timer > 1) //一秒后还停留在原地  {  switch (Random.Range(0, 4)) //随机方向弹开  {  case 0:  GetComponent<Rigidbody2D>().AddForce(transform.up \* 0.01f);  break;  case 1:  GetComponent<Rigidbody2D>().AddForce(-transform.up \* 0.01f);  break;  case 2:  GetComponent<Rigidbody2D>().AddForce(transform.right \* 0.01f);  break;  case 3:  GetComponent<Rigidbody2D>().AddForce(-transform.up \* 0.01f);  break;  }  }  }  }  //离开碰撞时调一次  private void OnCollisionExit2D(Collision2D collision)  {  timer = 0; //计时归零  }  private void Update()  {  transform.Rotate(0, 0, 0.0001f); //物体必须处于非完全静止状态,持续碰撞才会生效  switch (state) //判断小球当前状态  {  case BallState.Bore: //上膛阶段  GetComponent<Rigidbody2D>().gravityScale = 0; //重力变为0  break;  case BallState.Ready: //准备阶段  GetComponent<CircleCollider2D>().isTrigger = false; //关闭触发  GetComponent<Rigidbody2D>().Sleep(); //小球停止不动  break;  }  }  } |

1. 然后做一个小球的预制件，主要是用于碰到添加道具时创建一个新小球，因为此时小球应属于战斗状态，所以预制件小球组件设置如下：

****

### 创建几何体（Enemy）

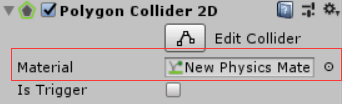
1. 随便创建几个几何体，这里我们先创建6个，都使用Text作为自己的子物体

****

1. 都添加上2D碰撞器，三角形、五边形、六边形、菱形使用PolygonCollider2D作为碰撞器，它的作用是根据精灵外型自行匹配碰撞器形状

****

1. 刚才不是创建了一个2D物理材质吗，现在添加到每个几何体的碰撞器内

****

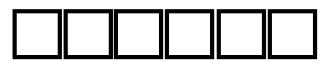
1. 创建一个几何体的脚本添加给每一个几何体，脚本内容如下：

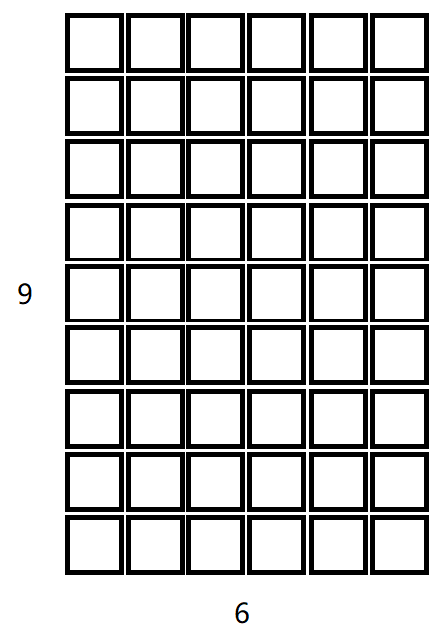
|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  /// <summary>  /// 挂每个几何体上  /// </summary>  public class Enemy : MonoBehaviour  {  Text number; //声明数字  private void Start()  {  number = GetComponentInChildren<Text>(); //找到子物体(数字)  }  private void Update()  {  if (Convert.ToInt32(number.text) < 1) //如果数字小于1时  Destroy(gameObject); //销毁几何体自身  }  } |

1. 几何体（Enemy）都创建好了，脚本也给它们挂上了，那就把它们做成预制件吧，创建关卡的时候再来生成实例

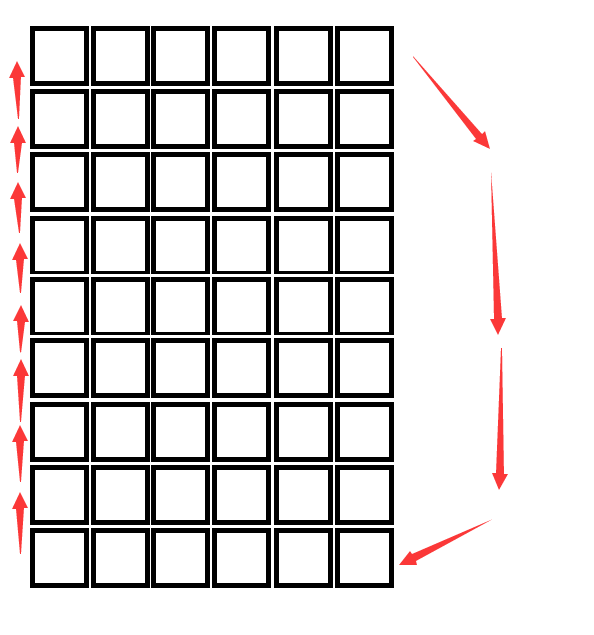
****

### 创建关卡

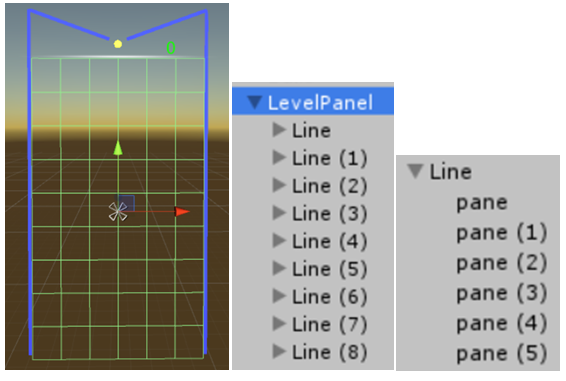
1. 几何体（Enemy）预制件都创建好了，现在我们来做关卡，关卡这一块相对就比较复杂了，我们每次从最下面一排生成物体，一排最多只能生成6个（仅限于本游戏中），那么我们可以把这6个位置看成6个格子，每个格子里随机生成物体  
   
2. 高度共有九层（仅限于本游戏中）

****

1. 当每过一回合，关卡下层会往上走一层，而最顶层又会回到最底层空白处

****

1. 在场景里创建6\*9个小格子来表示，并进行分层管理，游戏运行时每回合在底层随机生成一次物体，生成的物体为该层的子物体，随该层往上移动

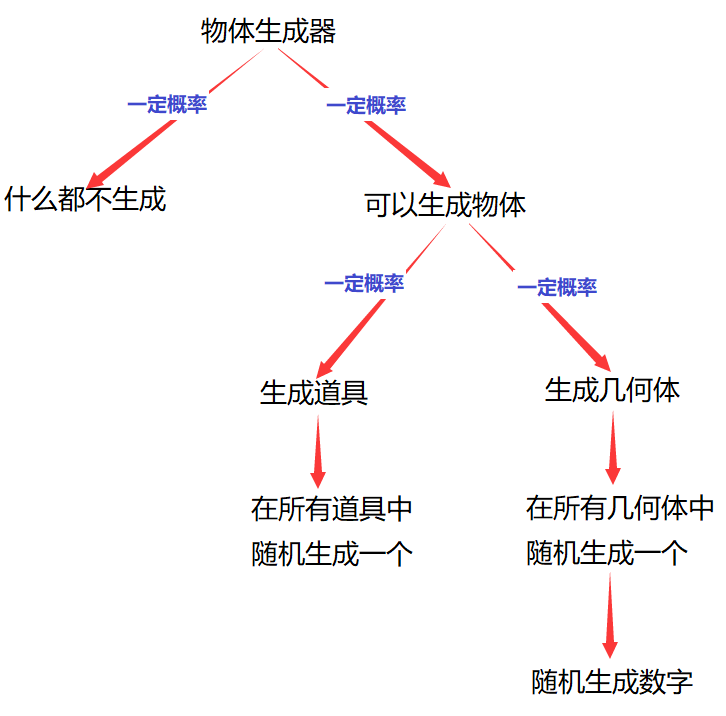
****

注意：为了让教程更加直观，上图中的绿色格子使用2D碰撞器来表示，实际上它们都是空物体，碰撞器没有任何用处，一定要移除掉，我们需要的只是小格子的位置，作为物体创建的生成点

1. 通过上面的描述，我们大概就知道关卡的运行方式了，接下来就是在脚本中实现，而在创建关卡运行脚本之前，我们先来创建一个关卡运行状态机：

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  /// <summary>  /// 关卡运行状态  /// </summary>  public enum LevelState  {  life, //运行中  pause, //暂停  die, //游戏结束  } |

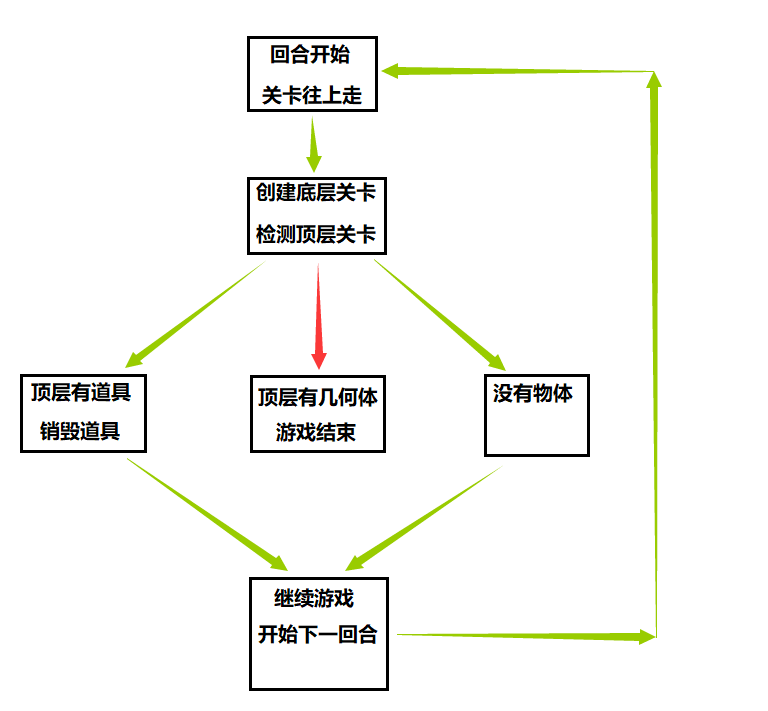
1. 接下来正式写关卡运行的脚本，关卡有两个功能，一个是生成物体，一个是往上升，我们先来写生成物体，在本项目中，生成物体需要遵循以下规则：

****

在脚本中表示：

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  /// <summary>  /// 挂LevelPanel上  /// </summary>  public class LevelCreate : MonoBehaviour  {  //在编辑器中将当前分数ScoreText拖进去  public Text scoreText;  //所有种类几何体,在编辑器把所有几何体预制件拖进去  public Transform[] Enemys;  //所有种类道具,在编辑器把所有道具预制件拖进去  public Transform[] stunts;  //物体生成器,决定格子里是否生成东西（几率可自行设定）  public Transform PaneFactory()  {  int chance = Random.Range(0, 4);  if (chance < 3) //75%不产生东西,  return null;  else //25%产生东西  return PaneManage();  }  //决定格子里该生产什么东西  Transform PaneManage()  {  int chance = Random.Range(0, 3);  if (chance < 2) //66%产生几何体  return CreateEnemy();  else //33%产生道具  return CreateStunt();  }  //随机生成道具  Transform CreateStunt()  {  //随机产生一个道具数组索引  int index = Random.Range(0, stunts.Length);  //生成该索引处道具  return Instantiate(stunts[index]);  }  //随机生成敌人  public Transform CreateEnemy()  {  //随机产生一个几何体数组索引  int index = Random.Range(0, Enemys.Length);  //生成该索引处几何体  Transform enemy = Instantiate(Enemys[index]);  //给几何体赋一个随机颜色  enemy.GetComponent<Renderer>().material.color = new Color(Random.value, Random.value, Random.value);  //给几何体一个随机旋转角度  enemy.rotation = Quaternion.Euler(0, 0, Random.Range(0, 90));  //获取几何体子物体数字的Transform组件  Transform tf = enemy.GetComponentInChildren<Text>().transform;  //子物体不旋转  tf.rotation = Quaternion.Euler(0, 0, 0);  //获取当前分数  int score = System.Convert.ToInt32(scoreText.text);  if (score < 100) //如果当前分数不超过100分  //几何体数字在 1~9 之间随机生成  enemy.GetComponentInChildren<Text>().text = Random.Range(1, 10).ToString();  else //当前分数超过100分  //几何体数字在 1~当前分数/10 之间随机生成  enemy.GetComponentInChildren<Text>().text = Random.Range(1, score / 10).ToString();  return enemy;  }  } |

1. 然后是关卡上升的脚本,规则如下：

****

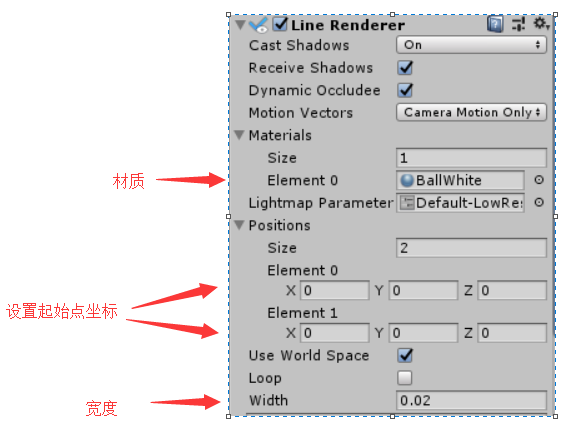
脚本里表示则为

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  /// <summary>  /// 挂LevelPanel上  /// </summary>  public class LevelMove : MonoBehaviour  {  //需要做一个菜单，游戏死亡时弹出，在编辑器把死亡菜单拖进来  public GameObject deathPanel;  //游戏初始状态为存活  public LevelState levelState = LevelState.life;  //声明一个关卡集合用来管理每层关卡  List<Transform> lineList = new List<Transform>();  LevelCreate levelcreate; //声明创建物体的类  private void Start()  {  levelcreate = GetComponent<LevelCreate>(); //获取创建关卡类  lineList = GetAllChild(transform); //获取第一层子物体(关卡)添加进关卡集合中  CreateLevel(); //游戏开始时创建一次底层的物体  }  private void Update()  {  //在游戏运行时按Esc  if (levelState == LevelState.life && Input.GetKeyDown(KeyCode.Escape))  {  levelState = LevelState.pause; //游戏状态变为暂停  deathPanel.SetActive(true); //启用菜单  Time.timeScale = 0; //游戏暂停  }  //在暂停状态时按Esc  else if (levelState == LevelState.pause && Input.GetKeyDown(KeyCode.Escape))  {  levelState = LevelState.life; //游戏状态变为运行  deathPanel.SetActive(false); //禁用菜单  Time.timeScale = 1; //游戏恢复  }  }  List<Transform> GetAllChild(Transform fatherObj) //获取所有第一层子物体  {  //声明一个集合放第一层所有子物体  List<Transform> sonList = new List<Transform>();  int number = fatherObj.childCount; //获取第一层子物体数量  for (int i = 0; i < number; i++)  {  //将所有第一层子物体添加进集合中  sonList.Add(fatherObj.GetChild(i));  }  return sonList; //返回第一层子物体集合  }  void CreateLevel() //创建底层关卡  {  Transform last = lineList[lineList.Count - 1]; //获取底层关卡,物体将从该层产生  List<Transform> sonList = GetAllChild(last); //获取底层所有小方格  //生成一个几何体（每次创建关卡至少有一个几何体）  Transform enemy = levelcreate.CreateEnemy();  int index = Random.Range(0, last.childCount); //随机定位一个格子  enemy.position = last.GetChild(index).position; //将几何体创建在该格子内  enemy.parent = last.GetChild(index); //几何体作为该格子的子物体可随关卡层移动  //然后在其它格子里随机生成物体  for (int i = 0; i < sonList.Count; i++)  {  if (i != index) //除了刚才已经有敌人的格子外  {  //声明一个变量接受生成的物体  Transform obj = levelcreate.PaneFactory();  if (obj != null) //如果成功生产出东西  {  obj.position = sonList[i].position; //将该东西生产在此方格  obj.parent = sonList[i]; //作为该方格的子物体随关卡层移动  }  }  }  }  //关卡往上走一层(第一层跳到最后)  public void LevelGetUp()  {  Vector3 tempPos = lineList[lineList.Count - 1].position; //获取最后层的坐标  //遍历所有关卡层  for (int i = lineList.Count - 1; i >= 0; i--)  {  if (i == 0) //如果是顶层  lineList[i].position = tempPos; //直接跳到底层  else //如果是其它层  lineList[i].position = lineList[i - 1].position; //移动到自己上一层  }  DestroyStunt(); //销毁顶层道具  lineList.Add(lineList[0]); //将第一层添加到集合最后  lineList.RemoveAt(0); //再移除第一层  CreateLevel(); //创建一次关卡关卡  if (Death()) //判断是否死亡  {  levelState = LevelState.die; //状态变为死亡  deathPanel.SetActive(true); //调用菜单  }  }  void DestroyStunt() //销毁顶层特技  {  //获取该层所有子物体  Transform[] lineSon = lineList[0].GetComponentsInChildren<Transform>();  for (int i = 0; i < lineSon.Length; i++) //遍历所有子物体的标签  {  if (lineSon[i].tag == "Stunt") //如果是道具  {  Destroy(lineSon[i].gameObject); //销毁该子物体  }  }  }  bool Death() //死亡判断  {  //获取顶层所有子物体  Transform[] lineSon = lineList[0].GetComponentsInChildren<Transform>();  for (int i = 0; i < lineSon.Length; i++) //遍历所有子物体的标签  {  if (lineSon[i].tag == "Enemy") //如果发现有几何体  return true; //直接游戏结束  }  return false; //如果一个都没有,游戏继续  }  } |

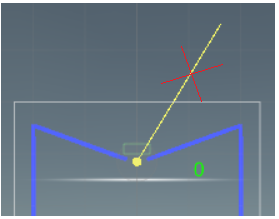
### 发射（小球从枪口往鼠标方向发射）

好的，敌人有了，小球有了，道具有了，关卡有了，我们可以来试试发射小球了。

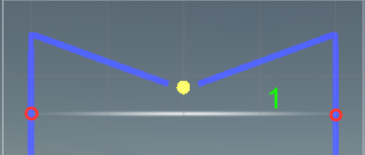
1. 首先制作瞄准线，瞄准线我们使用LineRenderer组件来实现，把它挂到枪口物体上

****

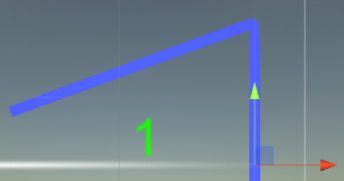
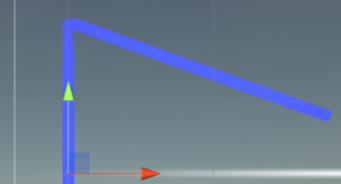
1. 设定小球发射范围，如果不设定范围，则会出现以下情况：

****

1. 所以我们需要在左右两边设定边界：

****

1. 边界用连个空物体来做就可以了，主要是获取它们的坐标来限制发射范围

****

****

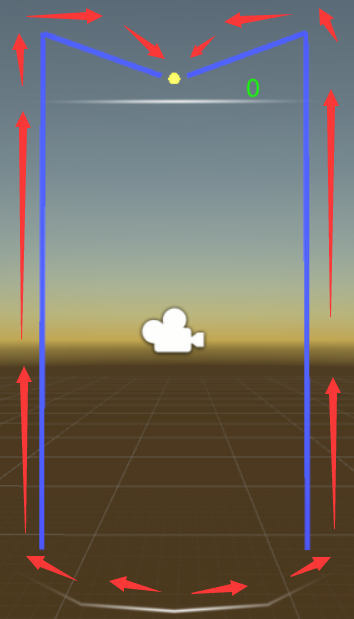
1. 小球的发射使用回合制，小球发射一次，关卡上升一层，结合着之前游戏运行状态机和小球状态机的思路，小球发射脚本可以这样来写，之后挂到枪口上：

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  public class Aim : MonoBehaviour //挂枪口Muzzle上  {  public GameObject balls; //把Balls拖进去  Rigidbody2D[] allBall; //声明一个数组用来管理所有小球  LineRenderer aimLine; //声明瞄准线  public Transform CriticalPointLeft; //把左边界拖进去  public Transform CriticalPointRight; //把右边界拖进去  public float shootingSpeed = 3.5f; //小球发射速度  public GameObject levelPanel; //把LevelPanel拖进去  bool levelStop; //判断关卡是否已上升  void Start()  {  Time.timeScale = 1; //游戏时间正常  allBall = balls.GetComponentsInChildren<Rigidbody2D>();//初始化(获取当前所有小球)  aimLine = GetComponent<LineRenderer>(); //获取枪口上的LineRenderer组件  }  void Update()  {  //当游戏状态为活着时  if (levelPanel.GetComponent<LevelMove>().levelState == LevelState.life)  {  if (Homing()) //所有小球都进入准备状态了  {  allBall = balls.GetComponentsInChildren<Rigidbody2D>(); //再次获取所有小球  if (levelStop) //关卡处于未上升状态  {  levelPanel.GetComponent<LevelMove>().LevelGetUp(); //调用关卡上升方法  levelStop = !levelStop; //关卡处于已上升状态  }  else  AimLaunch(); //关卡上升完成后可进行瞄准发射  }  }  }  //判断所有小球是否都进入准备状态  public bool Homing()  {  //发现任何小球不在准备状态都返回False  for (int i = 0; i < allBall.Length; i++)  {  if (allBall[i].GetComponent<BallMove>().state != BallState.Ready)  return false;  }  return true; //未发现不在准备状态的小球，返回True  }  void AimLaunch() //瞄准发射  {  if (Input.GetMouseButtonDown(0)) //点击鼠标左键  {  aimLine.SetPosition(0, transform.position); //在枪口处生成瞄准线起点  }  if (Input.GetMouseButton(0)) //按住鼠标左键不放  {  //获取鼠标坐标  Vector3 v = Camera.main.ScreenToWorldPoint(Input.mousePosition);  //限制瞄准范围  v = DirectionRestriction(v, CriticalPointLeft, CriticalPointRight);  //将被限制过的鼠标坐标实时给瞄准线结束点  aimLine.SetPosition(1, new Vector2(v.x, v.y));  }  if (Input.GetMouseButtonUp(0)) //抬起鼠标左键  {  StartCoroutine(LineLaunch(transform.position)); //启动协程发射小球  aimLine.SetPosition(1, transform.position); //让结束点和起点重合(撤销瞄准线)  levelStop = !levelStop; //关卡标记为可上升状态  }  }    IEnumerator LineLaunch(Vector3 muzzlePos) //用协程排队发射小球  {  Vector3 pos1 = aimLine.GetPosition(1);//获取瞄准线结束点坐标  Vector3 directionAttack = (pos1 - muzzlePos).normalized;//获取瞄准结束点与枪口的方向向量  for (int i = 0; i < allBall.Length; i++) //挨个发射小球  {  //被发射的小球变为战斗状态  allBall[i].GetComponent<BallMove>().state = BallState.Battle;  //球往瞄准结束点方向寻路移动  allBall[i].AddForce(directionAttack \* shootingSpeed \* Time.deltaTime);  yield return new WaitForSeconds(0.1f); //每隔0.1秒发射一个  }  }  //限定枪口瞄准方向  Vector3 DirectionRestriction(Vector3 v, Transform left, Transform right)  {  //最左不能左过左边界  if (v.x < left.position.x)  v.x = left.position.x;  //最右不能右过右边界  if (v.x > right.position.x)  v.x = right.position.x;  //高度不能超过边界  if (v.y > left.position.y)  v.y = left.position.y;  return v; //返回被限制后的坐标  }  } |

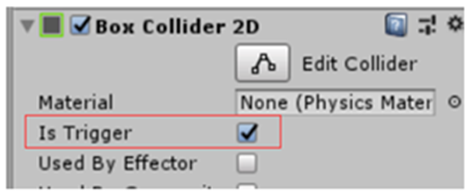
### 小球寻路（让发射的小球自动回到枪口）

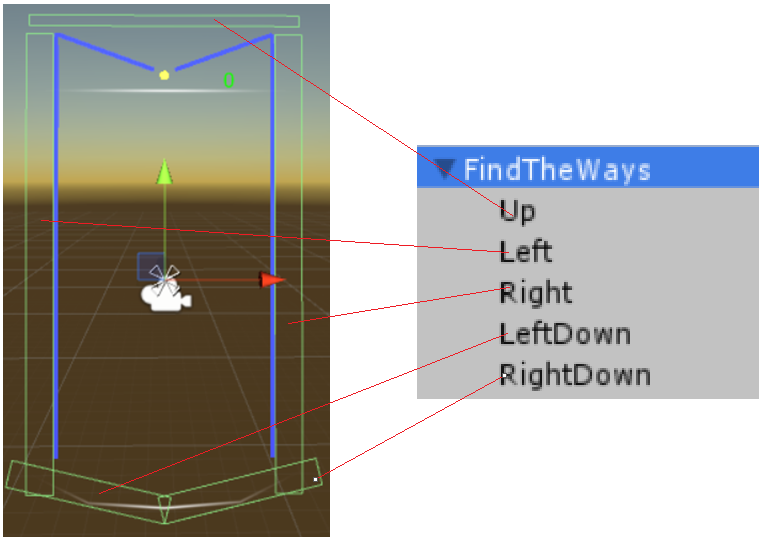
小球发射出去后怎么重新回到枪口位置呢，这是一个问题

1. 首先设定好小球回到枪口的路径

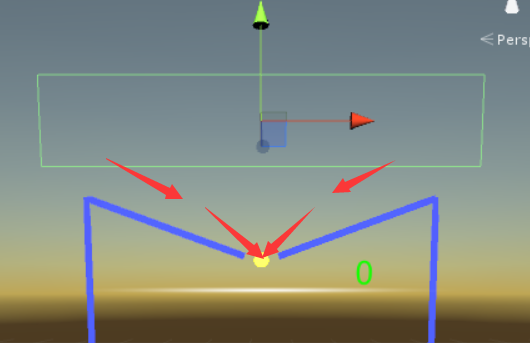
****

1. 在相应位置做好寻路触发器（做几个空物体，添加2D碰撞器，然后勾上isTrigger选项），做一个空物体FindTheWays来管理它们

****

****

1. 每个寻路触发器主要做一件事：让小球往某个方向移动，所以可以创建一个通用脚本挂每个寻路触发器上，而只有顶上的触发器需要让小球准确回到枪口位置

****

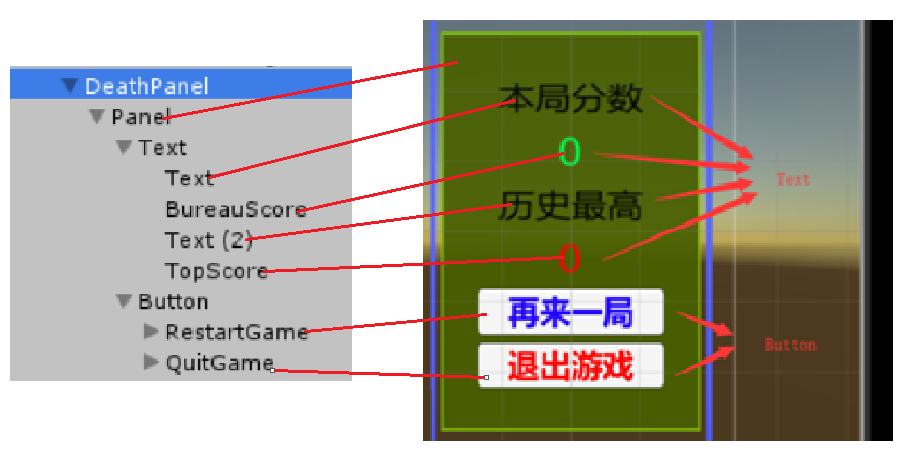
那么这个脚本可以这样写：

|  |
| --- |
| /// FindTheWays下的所有寻路碰撞器都挂一个  /// </summary>  public class BallFindWay : MonoBehaviour  {  public Transform muzzle; //在编辑器把枪口拖进去，我们需要枪口的坐标  public float boreSpeed=0.2f; //上膛速度  private void OnTriggerStay2D(Collider2D ball) //触发时持续调用  {  //获取小球的刚体  Rigidbody2D r2d = ball.GetComponent<Rigidbody2D>();  switch (name) //根据寻路碰撞器的名字决定施加力的方向  {  case "LeftDown":  r2d.AddForce(-transform.right \* 0.002f);  break;  case "RightDown":  r2d.AddForce(transform.right \* 0.003f);  break;  case "Left":  case "Right":  r2d.AddForce(transform.up \* 0.002f);  break;  case "Up":  //启动协程寻路（上膛）  StartCoroutine(MoveToMuzzle(ball.transform, muzzle));  //打开小球触发器,使小球能越过枪口阀门  ball.GetComponent<CircleCollider2D>().isTrigger = true;  break;  }  }  //使用协程寻路让小球朝枪口处移动  public IEnumerator MoveToMuzzle(Transform ball, Transform muzzle)  {  ball.GetComponent<BallMove>().state = BallState.Bore; //小球状态改为上膛状态  while (ball.GetComponent<BallMove>().state == BallState.Bore) //如果是上膛状态  {  //小球往枪口处寻路,完成上膛  ball.position = Vector3.MoveTowards(ball.position, muzzle.position, boreSpeed \* Time.deltaTime);  yield return new WaitForFixedUpdate(); //每次循环间隔1帧  //如果小球位置和枪口位置接近  if ((ball.position - muzzle.position).sqrMagnitude <= 0.001f)  {  ball.GetComponent<BallMove>().state = BallState.Ready; //小球进入准备阶段  ball.position = muzzle.position; //将小球定在枪口位置  }  }  }  } |

1. 完成以上的工作，游戏基本功能就算完成了，可是有没有感觉还少点什么，是的一般的游戏都会有一个菜单面板，那我们接下来做一个菜单面板：

### 菜单面板

1. 使用UGUI搭建一个

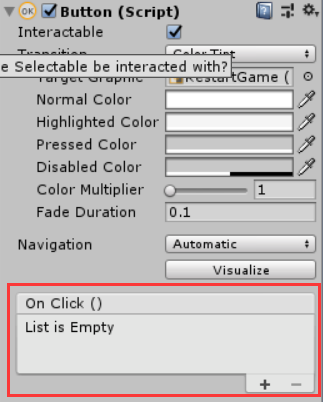
****

好看吗？我觉得还行。

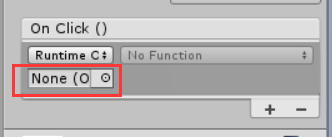
1. 创建脚本挂在DeathPanel上：

|  |
| --- |
| using UnityEngine;  using UnityEngine.SceneManagement;  using UnityEngine.UI;  /// <summary>  /// 挂DeathPanel上  /// </summary>  public class DeathBalance : MonoBehaviour  {  public Text score; //把当前分数ScoreText拖进去  public Text bureauScore; //把本局分数BureauScore拖进去  public Text topScore; //把最高分数TopScore拖进去  private void OnEnable()  {  bureauScore.text = score.text; //结算本局分数  if (PlayerPrefs.HasKey("分数")) //如果已经存储了分数  topScore.text = PlayerPrefs.GetString("分数"); //就获取上次存的最高分数  //让本局分数和最高分数比较，如果本局分数比最高分数大  if (Convert.ToInt32(bureauScore.text) > Convert.ToInt32(topScore.text))  {  topScore.text = bureauScore.text; //更新最高分数  PlayerPrefs.SetString("分数", bureauScore.text); //存储最高分数  }  }  public void RestartGame() //重新开始,挂按钮RestartGame上  {  SceneManager.LoadScene("ElasticBall"); //加载场景(提前保存一个场景)  }  public void QuitGame() //退出游戏,挂按钮QuitGame上  {  Application.Quit();  }  } |

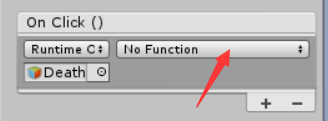
1. 如果我们点击“再来一局”按钮，就会执行脚本里的“RestartGame”方法，那么它们两个是怎么搭上线的呢？过程很简单，首先在编辑器里找到按钮这个物体，在它的Button组件里有一个添加事件面板

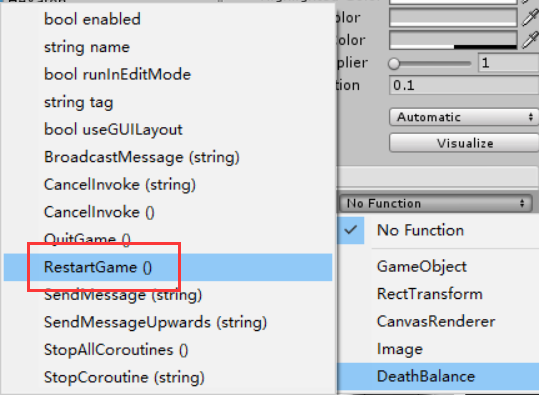
****

1. 点击加号+，把挂有这个脚本的物体（DeathPanel）这个小筐里：

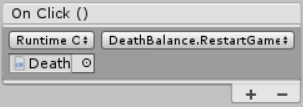
****

1. 然后在箭头所指位置寻找相应的脚本，继而找到相应方法：

****

****

1. 完成后，当我们点击该按钮，就会执行该方法了

****

1. 之前的一些脚本会调用到该菜单面板，如挂在LevelPanel上的LevelMove脚本，在编辑器里将菜单面板拖进去

### 总结

整个项目基本上算是完成了，游戏数值可以根据个人喜好自行设定，如生成图形的概率、数值等等。

弹球游戏看似是一个非常简单的小游戏，实际上实现起来，不仅用到了很多Unity的物理组件、组件的脚本操作，还对我们架构关卡、实现游戏机制带来了不小的挑战。相信通过这个示例教程，我们能对Unity的2D游戏开发有更深的认识，以及对2D物理系统能够更好的运用。