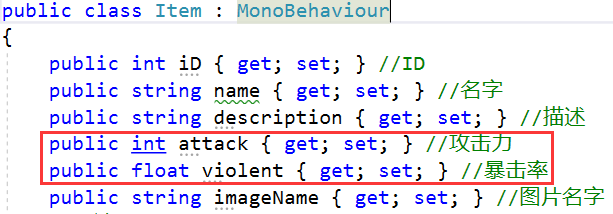
大家好,又是新的一期,这一期我们来制作背包物品的分类。

背包里的物品显然不会只有武器一种,而包含了很多种类,比如还有衣服,道具等待,并且,不同的物品具有的属性也不同,比如武器有攻击力没有防御力,衣服有防御里没有攻击力，先来丰富一下我们的Json数据配置表：

|  |
| --- |
| [  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*武器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  {  "ID": 1001,  "Name": "匕首",  "Description": "杀伤力较小,但方便携带。", //物品描述  "Attack": 10, //攻击力  "Violent": 0.05, //暴击率  "ImageName": "BiShou" //对应精灵图片名称  },  {  "ID": 1002,  "Name": "斧头",  "Description": "破坏力惊人,无奈重量较重,一般人无法使用。",  "Attack": 80,  "Violent": 0.3,  "ImageName": "FuTou"  },  {  "ID": 1003,  "Name": "弯刀",  "Description": "操作灵活,攻守兼备,且便于携带。",  "Attack": 30,  "Violent": 0.2,  "ImageName": "WanDao"  },  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*衣服\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  {  "ID": 2001,  "Name": "烈焰甲",  "Description": "具有较高的防高温属性,受伤时不容易找到伤口,耽误抢救。",  "PhysicalDefense": 60, //物理防御  "MagicDefense": 45, //魔法防御  "ImageName": "LieYanJia"  },  {  "ID": 2002,  "Name": "坚强之凯",  "Description": "因为它超凡脱俗的色系,穿上它的人都异常坚强。",  "PhysicalDefense": 80,  "MagicDefense": 20,  "ImageName": "JianQiangZhiKai"  },  {  "ID": 2003,  "Name": "圣诞纪念服",  "Description": "不能水洗也不能干洗更不能血洗,只能洗衣机洗。",  "PhysicalDefense": 5,  "MagicDefense": 2,  "ImageName": "ShengDanJiNianFu"  },  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*道具\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  {  "ID": 3001,  "Name": "青椒乌龟",  "Description": "百年的乌龟配上百年的青椒发酵百年而成。",  "AddHP": 300,  "AddMP": 200,  "ImageName": "QingJiaoWuGui"  },  {  "ID": 3002,  "Name": "黄焖海星",  "Description": "据说这道菜的发明者在做黄焖鸡的时候不小心把鸡放成了海星。",  "AddHP": 100,  "AddMP": 50,  "ImageName": "HuangMenHaiXing"  },  {  "ID": 3003,  "Name": "红烧章鱼",  "Description": "只有当海底火山爆发时才会被喷上岸,上岸时已经熟了。",  "AddHP": 150,  "AddMP": 20,  "ImageName": "HongShaoZhangYu"  },  {  "ID": 3004,  "Name": "白灼扇贝",  "Description": "白天灼烤而成,因为晚上厨师不上班",  "AddHP": 30,  "AddMP":10,  "ImageName": "BaiZhuoShanBei"  }  ] |

那么数据类Item显然已将不能满足我们的需求:



我们可以把Item数据类提升为一个基类,让其它的物品数据类去继承他,并且创建一个枚举来为不同物品设置类型:

|  |
| --- |
| public enum GoodsType  {  Weapon, //武器  Clothing, //衣服  Prop, //道具  }  public class Item  {  public GoodsType type;  public int iD { get; set; } //ID  public string name { get; set; } //名字  public string description { get; set; } //描述  public string imageName { get; set; } //图片名字  } |

创建一个武器数据类:

|  |
| --- |
| public class WeaponsItem : Item  {  public int attack { get; set; } //攻击力  public float violent { get; set; } //暴击率  public WeaponsItem(int \_iD, string \_name, string \_description, int \_attack, float \_violent, string \_imageName)  {  type = GoodsType.Weapon; //武器类型  iD = \_iD;  name = \_name;  description = \_description;  attack = \_attack;  violent = \_violent;  imageName = \_imageName;  }  } |

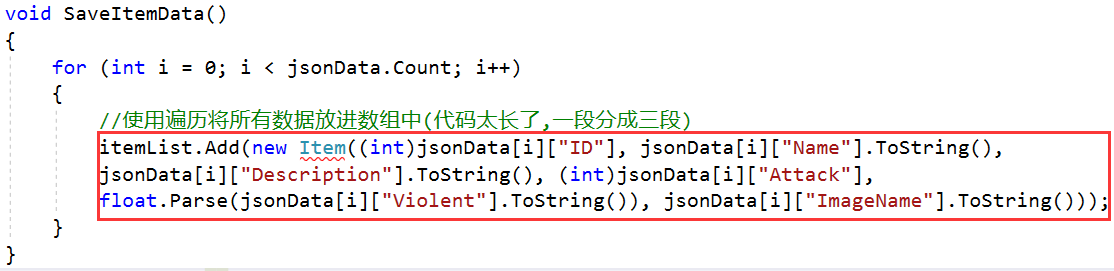
创建一个衣服数据类:

|  |
| --- |
| public class ClothingItem : Item  {  public int physicalDefense { get; set; } //物理防御  public int magicDefense { get; set; } //魔法防御  public ClothingItem(int \_iD, string \_name, string \_description, int \_physicalDefense, int \_magicDefense, string \_imageName)  {  type = GoodsType.Clothing; //衣服类型  iD = \_iD;  name = \_name;  description = \_description;  physicalDefense = \_physicalDefense;  magicDefense = \_magicDefense;  imageName = \_imageName;  }  } |

创建一个道具数据类:

|  |
| --- |
| public class PropItem : Item  {  public int addHP; //加血  public int addMP; //加蓝  public PropItem(int \_iD, string \_name, string \_description, int \_addHP, int \_addMP, string \_imageName)  {  type = GoodsType.Prop; //道具类型  iD = \_iD;  name = \_name;  description = \_description;  addHP = \_addHP;  addMP = \_addMP;  imageName = \_imageName;  }  } |

这样一来ItemData类里的存储数据方法也将不适用:



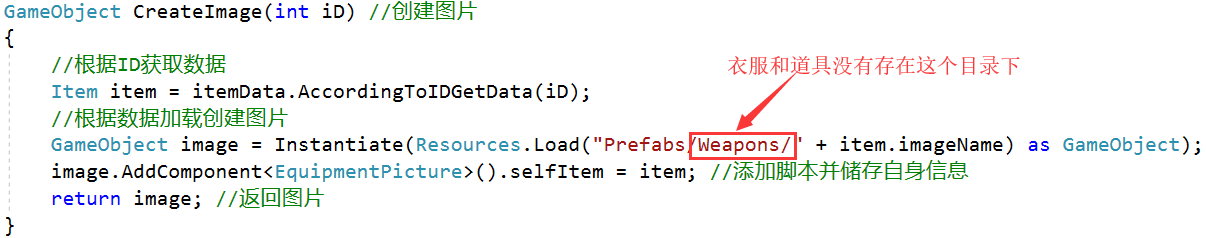
我们重构一下这个方法,把物品种类的多样性考虑进去,根据物品的ID第一个数字(武器是’1’,衣服是’2’,道具是’3’)来区分不同类型物品,然后分别实例化数据类:

|  |
| --- |
| void SaveItemData()  {  for (int i = 0; i < jsonData.Count; i++)  {  int iD = (int)jsonData[i]["ID"]; //获取当前数据的ID  switch (iD / 1000) //判断ID的第一位数字  {  case 1: //如果是1,将所有武器数据放进数组中  itemList.Add(new WeaponsItem((int)jsonData[i]["ID"], jsonData[i]["Name"].ToString(),  jsonData[i]["Description"].ToString(), (int)jsonData[i]["Attack"],  float.Parse(jsonData[i]["Violent"].ToString()), jsonData[i]["ImageName"].ToString()));  break;  case 2: //如果是2,将所有衣服数据放进数组中  itemList.Add(new ClothingItem ((int)jsonData[i]["ID"], jsonData[i]["Name"].ToString(),  jsonData[i]["Description"].ToString(), (int)jsonData[i]["PhysicalDefense"],  (int)jsonData[i]["MagicDefense"], jsonData[i]["ImageName"].ToString()));  break;  case 3: //如果是3,将所有道具数据放进数组中  itemList.Add(new PropItem((int)jsonData[i]["ID"], jsonData[i]["Name"].ToString(),  jsonData[i]["Description"].ToString(), (int)jsonData[i]["AddHP"],  (int)jsonData[i]["AddMP"], jsonData[i]["ImageName"].ToString()));  break;  }  }  } |

我们把衣服和道具的预制件也做好:



CreateBackpack类里创建图片的方法也需要做适当调整:



修改CreateBackpack类里的CreateImage方法:

|  |
| --- |
| public GameObject CreateImage(int iD) //创建图片  {  //根据ID获取数据  Item item = itemData.AccordingToIDGetData(iD);  GameObject image = null;  switch (item.type) //判断物品类型,去往不同路径加载预制件  {  case GoodsType.Weapon:  image = Instantiate(Resources.Load("Prefabs/Weapons/" + item.imageName) as GameObject);  break;  case GoodsType.Clothing:  image = Instantiate(Resources.Load("Prefabs/Clothings/" + item.imageName) as GameObject);  break;  case GoodsType.Prop:  image = Instantiate(Resources.Load("Prefabs/Props/" + item.imageName) as GameObject);  break;  }  image.AddComponent<EquipmentPicture>().selfItem = item;//添加脚本并储存自身信息  return image; //返回图片  } |

当这个创建图片的方法执行后,马上会有一个将它放进小格子的方法CreateImageInSlot,应先判断格子数量是否大于图片数量,只有这样才能顺利创建图片,并放入空的格子,所以要先获得小格数量和物品数量进行对比后,才能调用CreateImage方法,并把实例化的图片存放起来，我们之前已经创建了获取所有小格子的方法，也在Start（）里将小格子都存进了slots集合里，那么我们来重新构建CreateImageInSlot方法:

|  |
| --- |
| //声明集合存放所有已经实例化的物品图片  public List<GameObject> goodsList = new List<GameObject>();  public void CreateImageInSlot(int iD) //根据ID创建图片放进空格里  {  if (goodsList.Count < slots.Count) //当物品数量小于格子数量时  {  GameObject image = CreateImage(iD); //创建物品图片  goodsList.Add(image); //将创建的物品图片放进物品集合中  for (int i = 0; i < transform.childCount; i++) //遍历所有小方格  {  if (transform.GetChild(i).childCount == 0) //如果小方格没有子物体,表示该小方格为空  {  image.transform.SetParent(transform.GetChild(i)); //将图片放进该小方格中  image.transform.localPosition = Vector3.zero; //居中  return; //放进去后退出整个方法  }  }  }  print("物品栏已满");  } |

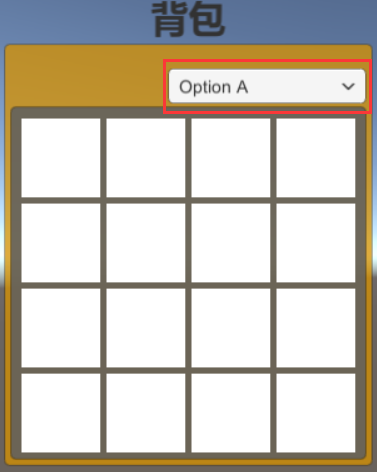
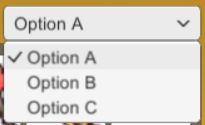
我们可以在Update里做一个测试,通过按大键盘上0~9不同的键来创建不同的装备:

|  |
| --- |
| void Update()  {  if (Input.GetKeyDown(KeyCode.Alpha0))  CreateImageInSlot(1001);  if (Input.GetKeyDown(KeyCode.Alpha1))  CreateImageInSlot(1002);  if (Input.GetKeyDown(KeyCode.Alpha2))  CreateImageInSlot(1003);  if (Input.GetKeyDown(KeyCode.Alpha3))  CreateImageInSlot(2001);  if (Input.GetKeyDown(KeyCode.Alpha4))  CreateImageInSlot(2002);  if (Input.GetKeyDown(KeyCode.Alpha5))  CreateImageInSlot(2003);  if (Input.GetKeyDown(KeyCode.Alpha6))  CreateImageInSlot(3001);  if (Input.GetKeyDown(KeyCode.Alpha7))  CreateImageInSlot(3002);  if (Input.GetKeyDown(KeyCode.Alpha8))  CreateImageInSlot(3003);  if (Input.GetKeyDown(KeyCode.Alpha9))  CreateImageInSlot(3004);  } |

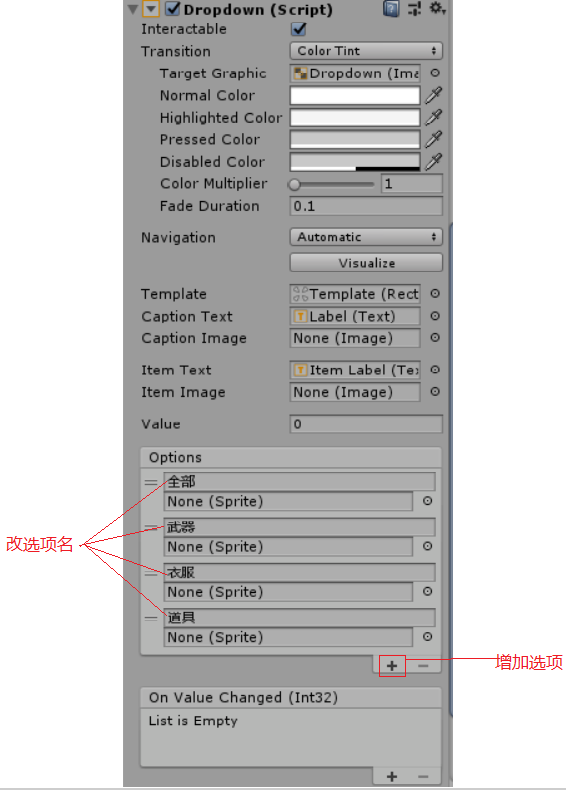


如果背包里的东西数量种类繁多，那我们在浏览时必定会眼花缭乱，如果将背包的物品进行分类，比如有一个专门显示武器的界面，专门显示衣服的界面，专门显示道具的界面，那么就会方便许多，下面就来实现这个效果：

首先在场景中创建一个下拉菜单Dropdown,作为BackpackPanel的子物体：

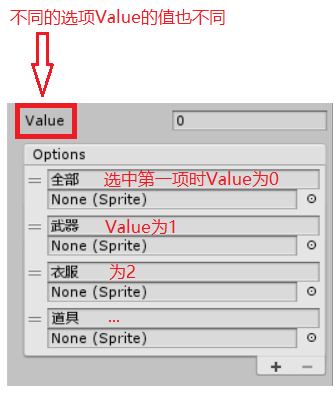
  

在组件里修改选项名字，并调整好大小：





在组件里有一个Value的值,会根据不同的选项显示不同的值：



组件下方有一个事件绑定窗口,点击+后可以绑定事件:



这个事件会在每次选项发生改变时执行,我们就可以做一个事件方法来绑定,方法里就写根据当前的Value值显示不同种类的物品,我们来做一个专门显示图片的脚本:

|  |
| --- |
| public class ShowGoods : MonoBehaviour //此脚本挂在BackpackPanel  {  EquipmentPicture[] allGoods; //存放当前所有物品图片  List<Transform> allSlot; //存放当前所有小格子  public void GetAllGoods() //拿到当前所有物品和小格子  {  allGoods = GameObject.Find("BackpackPanel").GetComponentsInChildren<EquipmentPicture>();  allSlot = FindObjectOfType<CreateBackpack>().slots;  }  } |

在脚本中创建一个有参方法,传入不同的类型就将该类型所有物品保存,再做一个方法把这些物品重新排列进小格子中:

|  |
| --- |
| void ClassifyShow(GoodsType type) //显示某种类型物品  {  List<EquipmentPicture> goodsList = new List<EquipmentPicture>(); //保存物品  foreach (var item in allGoods)  {  //将满足类型条件的物品保存  if (item.selfItem.type == type)  goodsList.Add(item); //保存  else //不满足的就移出小格子并禁用Image  {  item.transform.SetParent(transform);  item.GetComponent<Image>().enabled = false;  }  }  AdaptationSlot(goodsList.ToArray()); //将保存的物品依次放入小格子  }  void AdaptationSlot(EquipmentPicture[] allGoods) //将要显示的物品依次放入小格子  {  for (int i = 0; i < allGoods.Length; i++)  {  allGoods[i].GetComponent<Image>().enabled = true; //启用Image组件  allGoods[i].transform.SetParent(allSlot[i]); //依次放进小格子  allGoods[i].transform.localPosition = Vector3.zero; //居中  }  } |

然后创建方法调用它们:

|  |
| --- |
| //分类显示(参数直接将下拉菜单Dropdown拖进去)  public void Show(Dropdown dropdown)  {  GetAllGoods(); //拿到当前全部物品和小格子  switch (dropdown.value) //根据value判断当前选项  {  case 0: //显示全部  AdaptationSlot(allGoods);  break;  case 1: //只显示武器  ClassifyShow(GoodsType.Weapon);  break;  case 2: //只显示衣服  ClassifyShow(GoodsType.Clothing);  break;  case 3: //只显示道具  ClassifyShow(GoodsType.Prop);  break;  }  } |

这个方法也将绑定在Dropdown的事件方法中。

将脚本挂在BackPackPanel上：



然后进入下拉菜单Dropdown的组建中：



运行游戏先将所有装备创建出来，然后根据不同选择显示不同物品：

好的，这一期要做的功能就走完了,我们下期再继续。