接着我们上一期,在上一期中我们配置了Json文件,然后成功地通过代码将Json数据获取到了游戏中,并创建了相应图片,那么这一节我们来做背包中图片的拖拽等相关功能:



武器图片被成功加载进背包中

主要功能:

1. 拖拽图片
2. 放入小方格
3. 交换图片
4. 拖拽图片

这个功能我们到EquipmentPicture脚本里来做,因为这个脚本会被挂在图片上,为这个脚本添加拖拽有关的接口,并实现接口方法:

|  |
| --- |
| using UnityEngine.EventSystems; //调用拖拽接口需要引入的命名空间  public class EquipmentPicture : MonoBehaviour, IDragHandler  {  public Item selfItem; //用来存储属于自身的信息  public void OnDrag(PointerEventData eventData) //拖拽时持续调用  {  transform.position = eventData.position;  }  } |

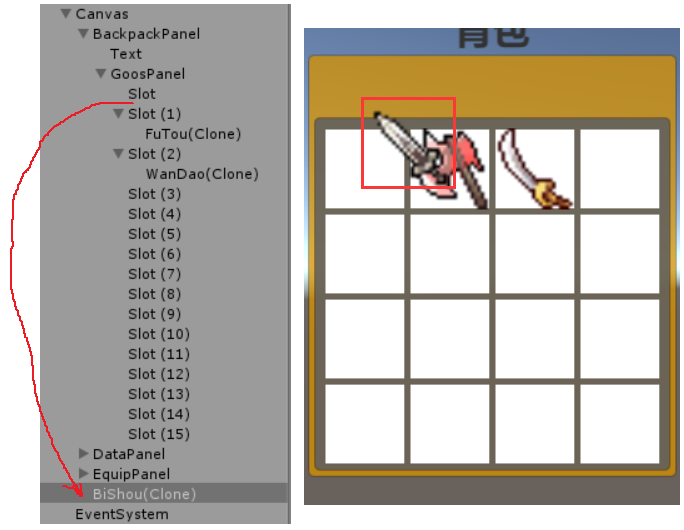
运行看下效果：



图片可以被拖拽

现在新的问题是，匕首图片被其它UI遮挡了，一般来说我们应该让被拖拽的图片凌驾于所有UI之上。

解决方法：在Unity的Hierarchy面板中，UI排列越往下，它的层级就越高，所以我们在拖拽时，将图片的排列位置放在最下，那就可以解决这个问题了：



我们在脚本里可以写拖拽时就将图片的排列顺序改变，然后再实现一个功能，就是拖拽结束后图片返回原位置，那么在拖拽之前，就要记录好原位置：

|  |
| --- |
| using UnityEngine.EventSystems; //调用拖拽接口需要引入的命名空间  public class EquipmentPicture : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler  {  public Item selfItem; //用来存储属于自身的信息  Transform original; //用于记录原位置  public void OnBeginDrag(PointerEventData eventData) //拖拽开始时调用一次  {  original = transform.parent; //记录原位置  //直接成为Canvas的子物体,排列在所有其它其物体之下  transform.SetParent(GameObject.Find("Canvas").transform);  }  public void OnDrag(PointerEventData eventData) //拖拽时持续调用  {  transform.position = eventData.position; //将鼠标的位置实时赋给图片  }  public void OnEndDrag(PointerEventData eventData) //拖拽结束时调用一次  {  transform.SetParent(original); //返回原位置  transform.localPosition = Vector3.zero; //居中  }  } |

现在鼠标就可以拖动图片正常移动了。

二．放入小方格

这里我的思路是，先找到所有小方格,当图片被拖拽进入某个小方格内,并且该小方格内没有其它图片(这里我们还没有做相同图片叠加的功能,所以逻辑暂时这样处理),则就把图片作为该小方格的子物体并居中。

1. 首先找到所有小方格，来到CreateBackpack里创建一个获取所有子物体小格子的方法：

|  |
| --- |
| public List<Transform> slots; //声明一个集合存放所有小格子  void GetAllSlot() //获取所有的GoodsSlot下所有的小格子  {  for (int i = 0; i < transform.childCount; i++)  {  slots.Add(transform.GetChild(i));  }  } |

在自身Start（）方法里调用。

1. 然后在EquipmentPicture脚本里创建一个放图片进小格子的方法:

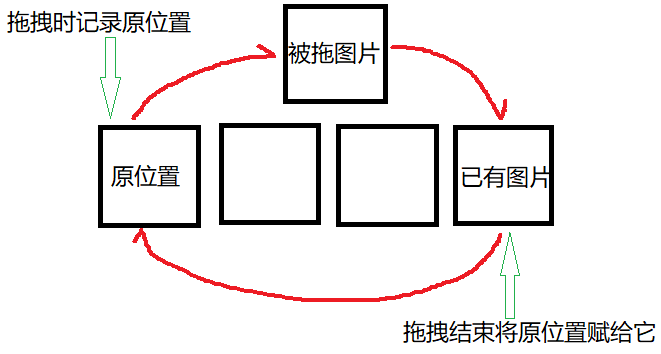
|  |
| --- |
| void GetIntoSlot() //放入小格子  {  //获取所有小格子  List<Transform> slots = FindObjectOfType<CreateBackpack>().slots;  for (int i = 0; i < slots.Count; i++)  {  //获取当前小格子  RectTransform slot = slots[i].GetComponent<RectTransform>();  //如果鼠标进入该小格子的范围并且小格子里没有东西  if (slot.rect.Contains(Input.mousePosition - slot.position) && slots[i].childCount == 0)  {  transform.SetParent(slots[i]); //将图片放入该小格子  transform.localPosition = Vector3.zero; //居中  return; //立刻结束该方法的调用  }  }  transform.SetParent(original); //返回原位置  transform.localPosition = Vector3.zero; //居中  } |

注意在这个方法内,把返回原处的代码提进来了,这个方法在拖拽结束时调用:

|  |
| --- |
| public void OnEndDrag(PointerEventData eventData) //拖拽结束时调用一次  {  GetIntoSlot();  } |

三.交换图片

交换图片的逻辑可以紧跟着将图片放入小格子的逻辑,在放入的逻辑中,我们是判断小格子内是否有别的图片,如果有则放不进去,现在可以稍微改一下:如果有则还是能放进去,但原先那张图片就要返回到被拖拽图片的原处:



GetIntoSlot方法需要做一些变动:

|  |
| --- |
| void GetIntoSlot() //放入小格子  {  //获取所有小格子  List<Transform> slots = FindObjectOfType<CreateBackpack>().slots;  for (int i = 0; i < slots.Count; i++)  {  //获取当前小格子  RectTransform slot = slots[i].GetComponent<RectTransform>();  //如果鼠标进入该小格子的范围  if (slot.rect.Contains(Input.mousePosition - slot.position))  {  //if (slot.rect.Contains(Input.mousePosition - slot.position) && slots[i].childCount == 0)  //{  // transform.SetParent(slots[i]); //将图片放入该小格子  // transform.localPosition = Vector3.zero; //居中  // return; //立刻结束该方法的调用  //}  //如果小格子里已经有图片,则该图片放入被拖图片的原位置并居中  if (slots[i].childCount > 0)  {  Transform oriP = slots[i].GetChild(0); //获取原图片  oriP.SetParent(original);  oriP.localPosition = Vector3.zero;  }  //将被拖图片放入该小格子并居中  transform.SetParent(slots[i]);  transform.localPosition = Vector3.zero;  return; //结束该方法的调用  }  }  transform.SetParent(original); //返回原位置  transform.localPosition = Vector3.zero; //居中  } |

代码中绿色被注释部分是刚才的写法,可以删除。

好的,这期的内容就到这里,下一期我们要来做将装备面板分为区,点击按钮切换的功能。