背包/仓库系统

重点部分：

1. 格子的交互逻辑
2. 背包/仓库数据更新存储逻辑

核心脚本：

1. InventoryManager.cs
2. UI_MainInventoryWindow.cs
3. UI_StorageBoxInventoryWindow.cs
4. StorageBox_Controller.cs
5. UI_InventoryWindowBase.cs
6. UI_ItemSlot.cs

InventoryManager

因为访问UI_xx需要调用UIManager的show方法，所以将一些对外接口封装起来放入InventoryManager中，在使用时直接调用单例进行使用

背包/仓库主要功能

物品快捷栏初始化

1. 从存档中读入保存的数据
2. 初始化每个格子和对应父窗口的信息（详情见下方<格子初始化>）
3. 如果有武器则将武器模型更新到角色模型身上

仓库初始化

1. 根据传入的`StorageBox_Controller storageBox, InventoryData inventoryData, Vector2Int windowSize`来初始化仓库
    a. StorageBox_Controller storageBox：
2. 设置数据和控制器
3. 设置窗口大小（根据预先测量的UI距离进行设置）
4. 生成窗口格子
    a. 遍历所有的仓库数据
    b. 从对象池中拿出格子进行实例化
    c. 初始每个格子中的物品数据及对应UI<格子初始化方法>
    d. 将格子数据添加到当前仓库数据列表中

仓库开启逻辑

1. 在输入管理器中会检查鼠标选中地图对象是否可以进行互动，由于仓库继承自`BuildingBase`并实现了`OnSelect`的方法，则可以在鼠标点击后打开仓库

仓库关闭逻辑

1. 玩家原理仓库时关闭仓库UI界面
    a. 检测当前场景中是否有玩家，如果存在玩家并且玩家与仓库之间的距离小于阈值则关闭仓库
2. 点击关闭按钮关闭仓库（在closeButton上绑定一个事件）
    a. 将格子中的数据放入对象池并清空所有格子

获取某个物品数量

1. 根据物品对应的configId变量当前背包/仓库格子，如果当前格子非空并且物品id一直则加上物品数量
2. 返回物品数量累计结果

获取空格子

1. 遍历当前背包/仓库格子，如果当前格子为空则返回对应的索引，否则返回-1代表背包/仓库满了

设置格子中的内容

1. 将传入的物品数据设置到传入索引对应的背包/仓库格子数据里
2. 调用格子初始化方法更新UI，详情见<格子初始化>

移除格子中的内容

1. 将传入的索引对应的格子数据设置为null
2. 调用格子初始化方法将UI更新为默认状态，详情见<格子初始化>

丢掉格子中的一件物品

1. 获得当前传入索引处的物品数据
2. 如果物品是武器需要在<移除格子中的内容>后使用Player_Controller中的ChangeWeapon将武器换成空手
3. 物品数据count -= 1，如果count == 0则<移除格子中的内容>，否则则<刷新格子UI信息>

获得某件物品的数量

1. 遍历背包/仓库中的全部格子
    a. 如果当前格子不为空并且与传入configId一致则查看当前物品是否为可堆叠物品
        i. 如果是可堆叠物品则 += 所有的物品数量
        ii. 否则总数+1

检查和添加物品到空格子

1. 得到空格子的索引，详情见<获取空格子>

2. 如果当前索引合法则在对应索引处创建一个物品，详情见<设置格子中的内容>

检测并堆放物体到格子上

1. 遍历当前背包/仓库中的格子
    a. 如果当前格子不为空并且与传入物品configId一致则检查当前格子是否达到堆叠上限，如果未达到堆叠上限则将物品放入该格子并更新UI数值
    b. 问题：当前可堆叠的物品可能是消耗品或者材料，如何才能方便获取堆叠？有两种方案 1. 可以在这两个类型中同时加入一个变量Count; 2. 抽象出一个可堆叠物品类型作为基类

添加物品

1. 根据传入的物品configId获取物品配置
2. 如果传入物品类型是武器则直接将武器<检查和添加物品到空格子>
3. 如果传入的物品类型是可堆叠的消耗品和材料，则优先检查是否能堆叠<检测并堆放物体到格子上>，如果不能则再找一个空格子放入<检查和添加物品到空格子>

使用物品（鼠标右键）

1. 如果玩家当前无法使用物品（canUseItem == false）则返回无法使用物品音效并退出
2. 如果当前传入index代表武器栏物品（右键点击物品代表卸下物品）
    a. <获取空格子>后如果空格子下标合法则交换武器格子和空格子中的数据并刷新UI
3. 如果当前传入index代表普通物品格子
    a. 格子中是武器
        i. 直接<交换物品槽中的数据并刷新UI>
    b. 格子中是消耗品
        i. 当前消耗品能否恢复生命值和饱食度，如果能恢复则调用Player_Controller中的对应方法恢复数据并刷新UI
        ii. 更新当前消耗品数量数据并刷新UI，当数量=0需要<移除格子中的内容>

c. 格子中是材料

i. 材料无法使用，需要返回错误音效进行播放

使用合成/建造功能时更新物品数量

1. 获取当前合成/建造配置中所包含的所有物品id和数量
2. 遍历当前物品快捷栏
   a. 如果当前格子中的数据不为空并且为目标物品
      i. 如果当前物品是可堆叠物品（材料/消耗品）
         1. 如果当前物品数量>所需物品数量，则更新格子中剩余物品数量并<刷新格子UI信息>
         2. 如果当前物品数量<所需物品数量，则更新合成/建造物品所需数量并<移除格子中的内容>
      ii. 如果当前物品不是可堆叠物品
         1. 物品数量-1后<移除格子中的内容>

使用合成/建造功能时更新物品数量

1. 遍历合成/建造配置表中所需的配置内容，<使用合成/建造功能时更新物品数量>

玩家使用武器攻击后更新武器栏状态

1. 如果当前玩家未装备武器则退出
2. 获取当前武器栏中的装备数据和装备配置
3. 当前装备数据中耐久度-=装备配置中单次攻击耐久度消耗值
   a. 如果装备数据中耐久度=0则认为武器已损坏，此时需要<移除格子中的内容>，并将更新玩家拿的武器模型
   b. 否则则<刷新格子UI信息>

格子主要功能

注意：

1. 格子UI：格子背景、默认图标、物品图标、物品数值、父窗口（宿主窗口）
2. 格子事件：OnMouseEnter（鼠标进入）、OnMouseExit（鼠标退出）、OnBeginDrag（鼠标开始拖拽）、OnDrag（鼠标拖拽中）、OnEndDrag（鼠标拖拽结束）、BindMouseEffect（鼠标进入格子后放大效果，退出后恢复大小）
3. 关键标志：isMouseStay、currentMouseEnterSlot（用于后面鼠标拖拽逻辑判断）

格子初始化方法

1. `Init`设置格子索引值、对应的父窗口（可能是仓库或者物品快捷栏）和格子的UI背景
2. `InitData` 通过传入的格子数据更新当前格子UI效果，空格子时恢复默认UI效果，格子中有物品时打开物品图标并设置不同的数值效果，例如武器是百分比，消耗品和材料都是个数，详情见<格子刷新UI信息>

刷新格子UI信息

1. 判断当前格子中数据类型`itemData.config.itemType`
    a. 武器：UI数字更新为当前durability + “%”
    b. 消耗品/材料：UI数字更新为当前count

检查鼠标右键是否可以使用物品（Update）

1. 如果当前为空格子或者没有对应使用方法（onUseAction）则返回
2. 如果鼠标停留并按下右键则使用对应方法（onUseAction）并播放音效

交换物品槽中的数据并刷新UI

1. 获得两个格子中的数据
2. 通过slot1.ownerWindow将slot1对应的数据设置为slot2的数据，详情见<设置格子中的内容>，slot2同理

重置格子图标

1. 重设格子图标父物体
2. localScale = vector3.one、localPosition = vector3.zero

鼠标进入事件

1. 将鼠标图像设置为手，将格子背景图片设置为选中框
2. 记录当前鼠标进入格子，并将isMouseStay设置为true

鼠标退出事件

1. 将鼠标图像和格子背景图片设置为普通形式
2. 重置当前鼠标进入格子数据，并将isMouseStay设置为false

鼠标开始拖拽事件

1. 如果当前格子中没有物品则停止拖拽
2. 将拖拽图标（SetParent）设置到DragLayer上保证不会被其他UI图层覆盖

鼠标拖拽事件

1. 如果当前格子中没有物品则停止拖拽
2. 将鼠标图像变为Handle
3. 更新拖拽图片的position为鼠标坐标

```
PointerEventData eventData
iconTransform.position = eventData.position;
```

鼠标停止拖拽事件

1. 如果格子中没有物品则停止拖拽
2. 如果拖拽到合法建筑物上（例如将木柴拖拽到火堆），判断逻辑见<检查当停止拖拽格子上的物品时是否点击到了建筑物身上>（InputManager.CheckSlotEndDragOnBuilding）
    a. 如果当前物品可以使用，则<重置格子图标>，恢复鼠标状态
3. 如果没有拖拽到其他格子里
    a. <重置格子图标>，恢复鼠标状态
    b. 如果拖拽到UI物体上或者是拖拽到了大型物体上则无视
    c. 如果鼠标未点击到地面直接无视
    d. 否则将物品扔到地上
        i. 根据物品configId拿到预制体等信息利用MapManager生成地图对象 `MapManager.Instance.GenerateMapObject(itemData.config.mapObjectConfigId, worldPosition, false);`
        ii. <丢掉格子中的一件物品>更新数据并刷新UI
4. 如果拖拽到其他格子里
    a. 拖拽到格子后的图标需要复原，并且恢复鼠标状态
    b. 如果拖拽到自己原本的格子中不处理
    c. 如果拖拽到武器格子需要查看拖拽的物品类型是否是武器，如果是武器则进行交换
    d. 如果拖拽到普通格子则直接<交换物品槽中的数据并刷新UI>
5. 存档管理器更新背包/仓库相关数据

常见问题：通过常见问题串联起模块执行顺序

1. 玩家捡到物品后如何添加到背包/仓库里？
    a. 左键点击地面物品后触发输入管理器 `CheckSelectMapObject` 检查逻辑
        i. 使用相机射线检测后将 `Physics.Raycast` out出来的hitInfo传给Player_Controller处理
    b. Player_Controller `OnSelectMapObject` 会根据传入的hitInfo查看对象是否能拾取

i. 不能拾取的条件情况有：1. 物品距离玩家过远；2. 鼠标未按下左键；3. 物品未配置拾取标记；4. 背包已满

c. 调用`InventoryManager.Instance.AddMainInventoryWindowItem(pickUpItemConfigId)`将拾取的物品放入物品快捷栏（背包）并销毁地图对象

2. 玩家扔掉物品后背包/仓库里的数据应该如何更新？地面UI如何更新？
   a. 左键按住拖拽物品到地面时会触发(UI_ItemSlot.cs)<鼠标开始拖拽事件>进行拖拽
   b. <鼠标拖拽事件>开始拖拽时需要更改鼠标指针形态，注意在拖拽过程中不断同步被拖拽物品图标的position以实现图标跟随指针的效果
3. 玩家交换物品时的逻辑是什么？
   a. 在鼠标拖拽完毕后交换两个格子中的数据并刷新UI
4. 玩家使用物品时的逻辑时什么？
   a. 详情见<使用物品（鼠标右键）>
5. 游戏结束后背包/仓库中的数据怎么保存？
   a. 在初始化背包/仓库时添加一个`OnGameSave`的事件，当游戏退出时调用所有的`OnGameSave`进行保存
6. 继续游戏时如何恢复背包/仓库中的数据？
   a. 从存档管理器中读出数据后设置背包/仓库中包含的格子数据和UI以及父窗口的关系，详情见<物品快捷栏初始化>和<仓库初始化>
7. 玩家合成/建造时如何消耗物品？
   a. 遍历合成/建造配置列表，再去遍历当前物品快捷栏，如果所需物品在物品快捷栏中需要查看当前物品是否是可堆叠物品还是其他物品，然后消耗物品数量更新UI信息，需要注意如果剩余物品数量≤0时需要移除对应的物品，详情见<使用合成/建造功能时更新物品数量>
8. 什么状态下触发背包/仓库的存储机制？
   a. 在游戏过程中背包/仓库数据一直在变动，只有当游戏结束的时候需要存储背包/仓库数据

背包/仓库数据结构

```
// 通用物品栏格子
[Serializable]
public class InventoryData
{
    // 物品栏中装的物品
    public ItemData[] itemDatas { get; protected set; }

    public InventoryData(int itemCount) {
        itemDatas = new ItemData[itemCount];
    }

    // 移除某一个物品
    public void RemoveItem(int index) {
        itemDatas[index] = null;
    }

    // 放置某一个物品
    public void SetItem(int index, ItemData itemData) {
        itemDatas[index] = itemData;
    }

}

// 物品快捷栏数据
[Serializable]
public class MainInventoryData : InventoryData
{
    public ItemData weaponSlotItemData { get; private set; }

    // 物品快捷栏构造函数
    public MainInventoryData(int itemCount) : base(itemCount) {}

    // 移除武器
    public void RemoveWeaponItem() {
        weaponSlotItemData = null;
    }

    // 放置武器
    public void SetWeaponItem(ItemData itemData) {
        weaponSlotItemData = itemData;
    }
}
```