# 背包系统制作

1. 技术概括

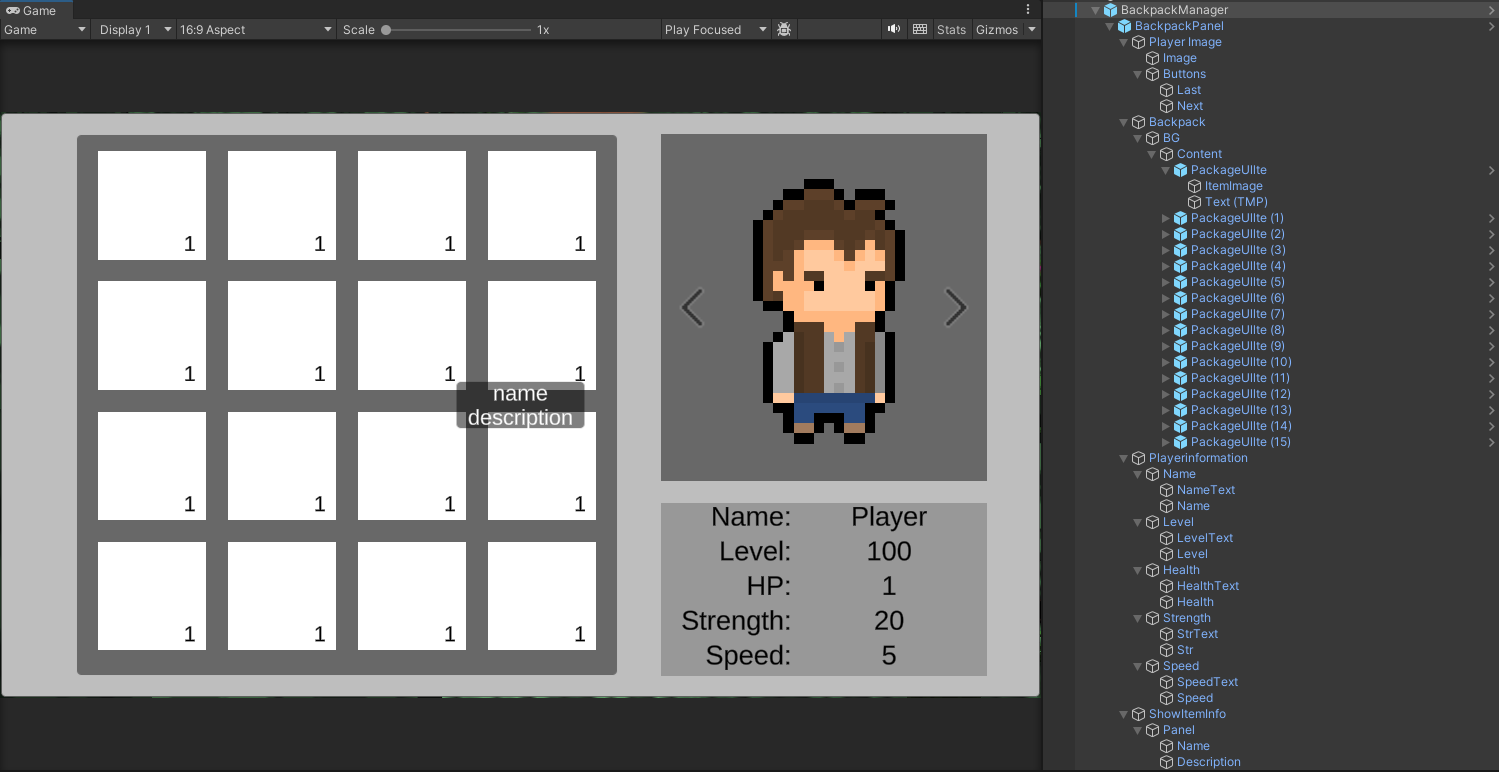
在Unity引擎中，ScriptableObject是一种高效的数据容器，专用于存储和管理游戏配置数据，与依赖游戏对象的MonoBehaviour不同，ScriptableObject以独立资源文件形式存在于项目中，无需绑定特定实体，这一特性使其成为构建背包系统的理想技术方案。

其核心优势在于数据与逻辑的彻底分离，开发者可通过预定义物品属性（如名称、图标、描述、堆叠上限等），在Unity编辑器中直接创建和调整数据资源。这种设计允许策划或美术人员独立参与内容制作，通过可视化界面修改物品参数，显著降低对程序代码的依赖，提升团队协作效率。

在背包系统实现中，ScriptableObject通常用于存储物品的静态模板数据（如基础属性），而动态运行时数据（如物品槽的实时堆叠数量）则由独立的库存管理系统维护。此外，ScriptableObject以资源共享模式运行，支持多个系统（如商店、任务奖励等）复用同一套物品数据集，避免冗余存储，确保数据一致性。

1. GUI图形界面

首先创建一个名为BackpackManager的背包管理画布，其整体层级如图：



背包系统的图形界面基于名为BackpackManager的画布构建，其层级结构如下：

* PlayerImage：展示玩家角色形象。
* Buttons：当存在多个角色时，提供切换角色信息的功能。
* Content：通过GridLayoutGroup组件自动管理物品图标的网格化布局。
* PlayerInformation：实时显示当前角色的属性数值（如生命值、攻击力等）。
* ShowItemInfo：悬浮式信息面板，用于展示物品的详细描述（如名称、效果说明等）。

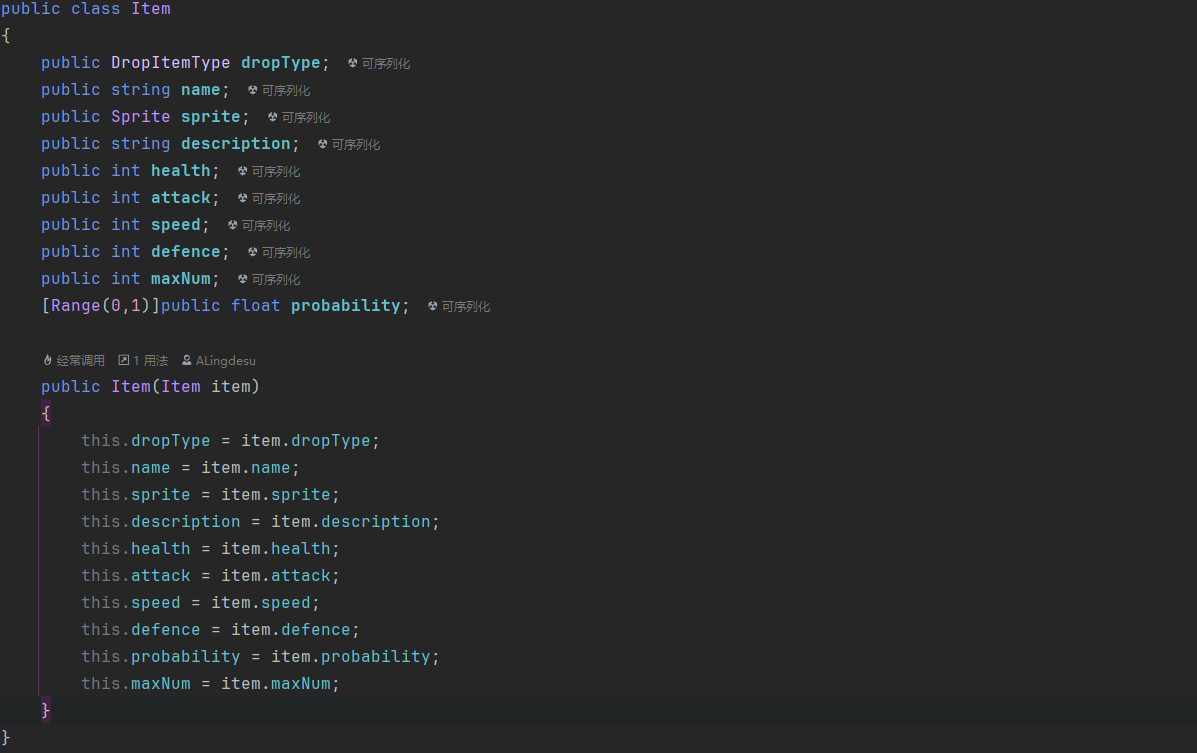
1. 物品系统设计
2. 编写物品类脚本
3. DropItemType掉落物品枚举类型

通过枚举类型（如DropItemType.Star）为每个物品分配唯一标识，避免字符串拼写错误导致的数据异常。枚举预设四种基础道具类型，确保类型定义的扩展性与安全性：



1. Item物品类

定义Item类，封装物品的核心属性（类型、名称、图标等），并通过构造函数实现实例化逻辑，支持动态生成物品副本:



1. 编写ScriptableObject数据库脚本
2. ItemInfo物品信息类

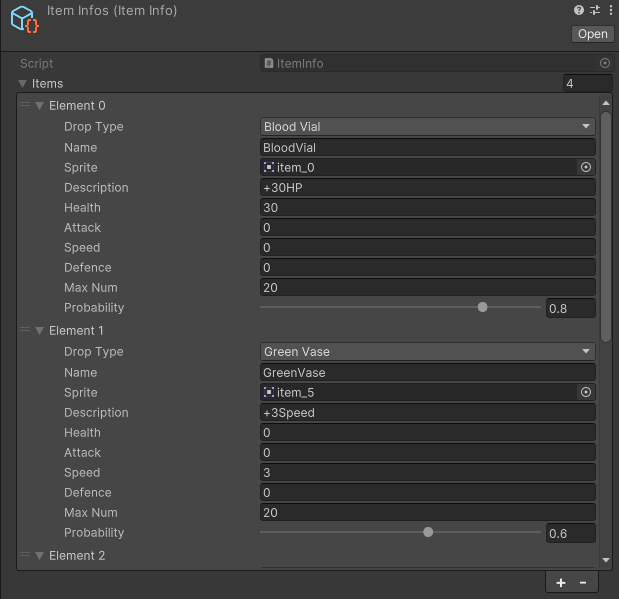
创建继承自ScriptableObject的ItemInfo类，通过公开的List<Item>属性集中管理所有物品数据，并提供以下关键方法：

* GetItemByType()：根据枚举类型检索物品。
* GetProbabilityByType()：获取指定物品的掉落概率。
* GetItemMaxCount()：查询物品的最大堆叠上限。



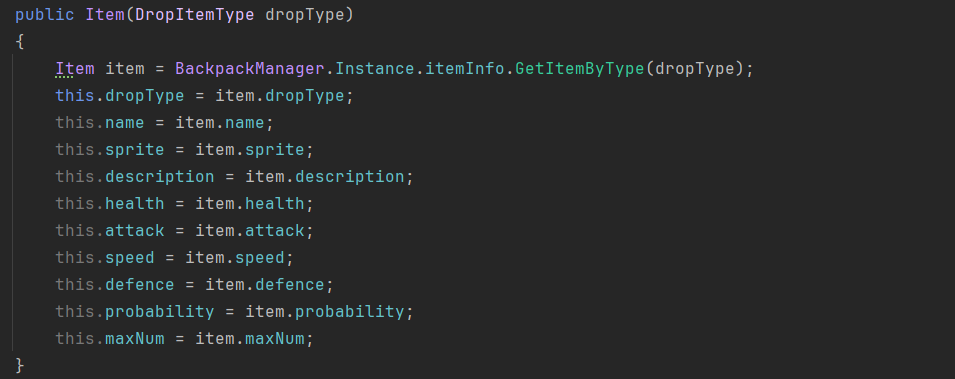
1. 创建相关数据库

在项目资产文件夹中创建ItemInfos文件并存储物品的相关数据。



1. 补充Item类的构造函数

添加一个通过物品类型创建Item实例的构造方法。



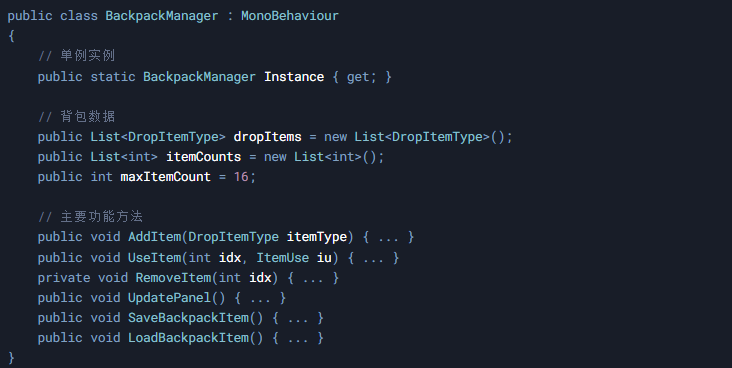
1. 背包系统设计
2. BackpackManager背包核心逻辑

通过单例模式实现全局访问，核心成员包括：

* dropItems：存储背包内物品类型的列表。
* itemCounts：记录每个物品槽的当前堆叠数量。
* maxItemCount：定义背包容量上限。

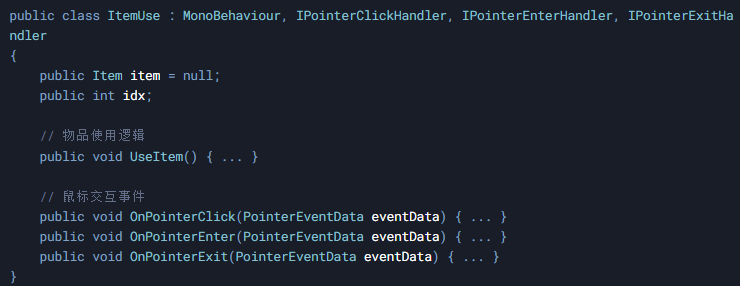
主要功能方法：

* AddItem()：根据类型添加物品，自动处理堆叠逻辑。
* UseItem()：通过索引使用物品，并触发数据更新。
* RemoveItem()：移除指定槽位的物品。
* UpdatePanel()：刷新背包UI显示状态。
* SaveBackpackItem()与LoadBackpackItem()：实现本地化数据存读。

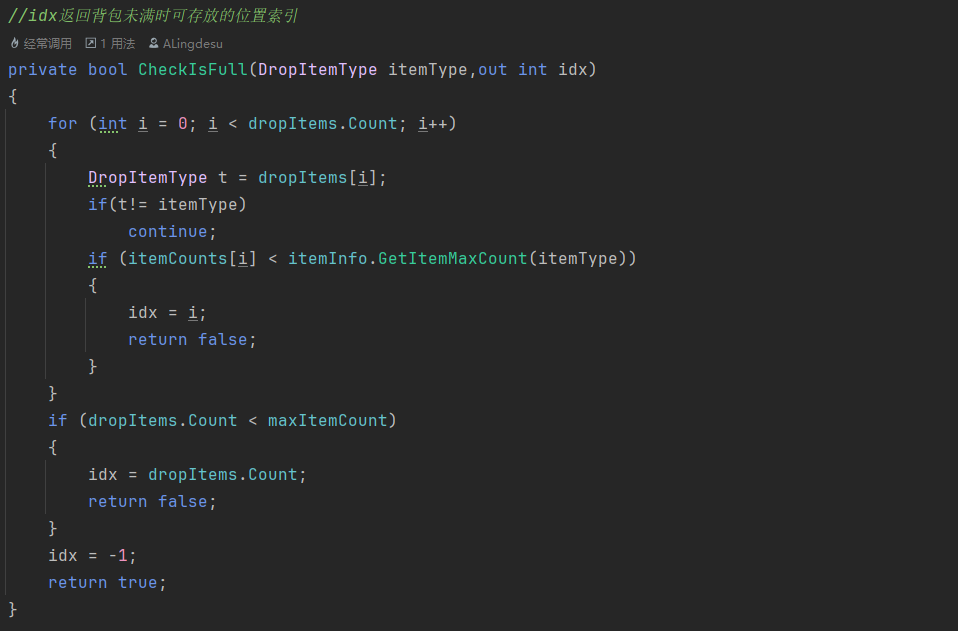


1. ItemUse物品交互逻辑

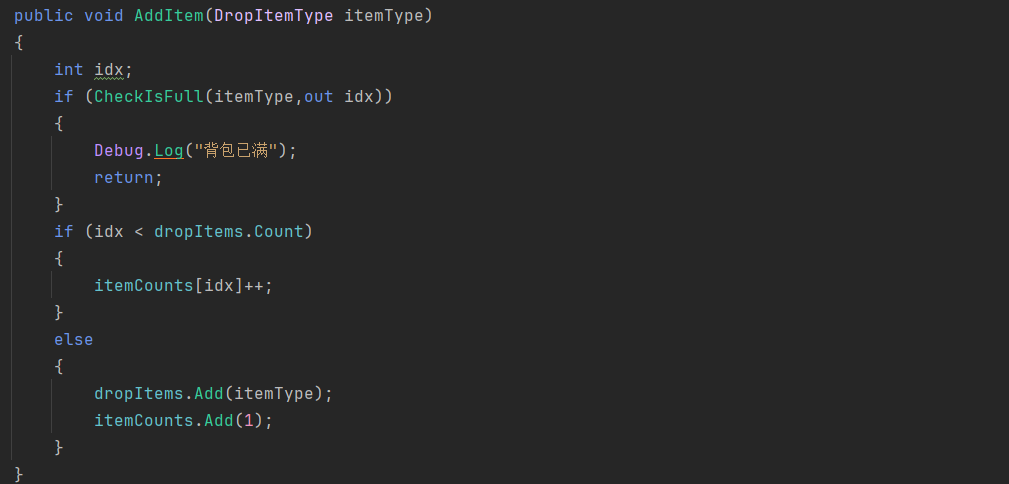
创建ItemUse脚本并挂载到每个背包格子物体上，并继承三个鼠标事件接口（鼠标点击、鼠标进入、鼠标离开），该脚本成员变量包括item物品和idx当前格子在背包栏中的索引，成员方法包括UseItem使用物品以及三个鼠标事件。



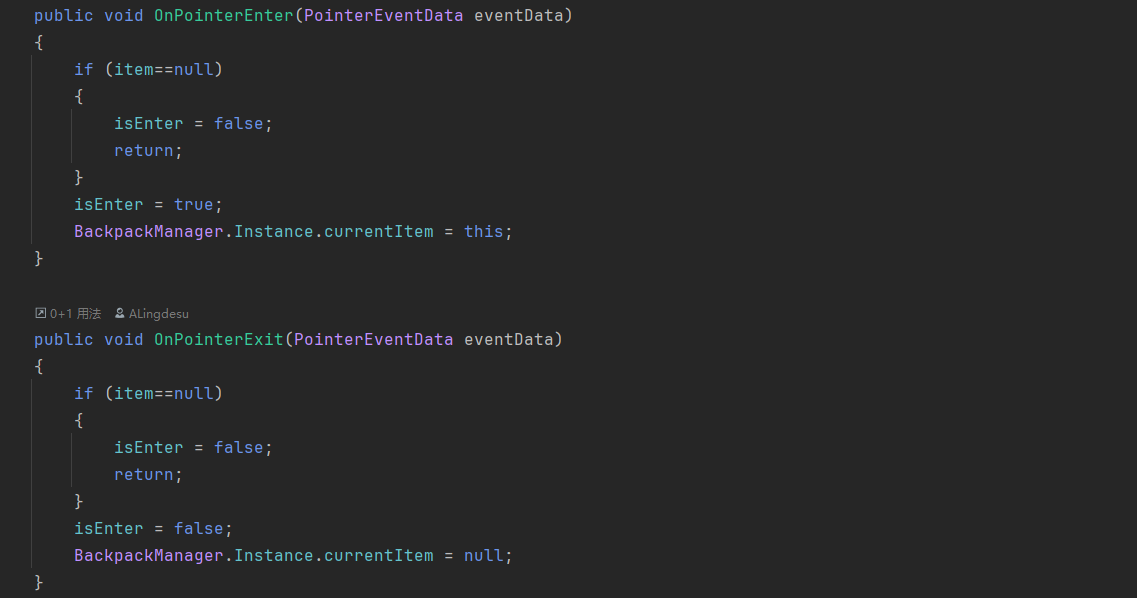
1. 核心功能逻辑
2. 物品添加逻辑
   1. 检查背包状态：通过CheckIsFull(DropItemType itemType,out int idx)函数来判断当前背包是否未满，若背包已满且添加的物品类型所属的物品也都达到最大上限则会返回true，反之则会返回false并且输出当前未满的格子索引，代码如下：



* 1. 添加物品：先判断CheckIsFull，如果返回false则通过idx添加物品，若idx小于dropItems的数量，说明列表中包含未达物品最大上限的同类型物品，则只需要对当前物品数量+1即可；反之则说明背包内无同类型物品或者同类型物品都达到堆叠最大值，则直接在dropItems添加当前类型，并对itemCounts添加1，代码如下：



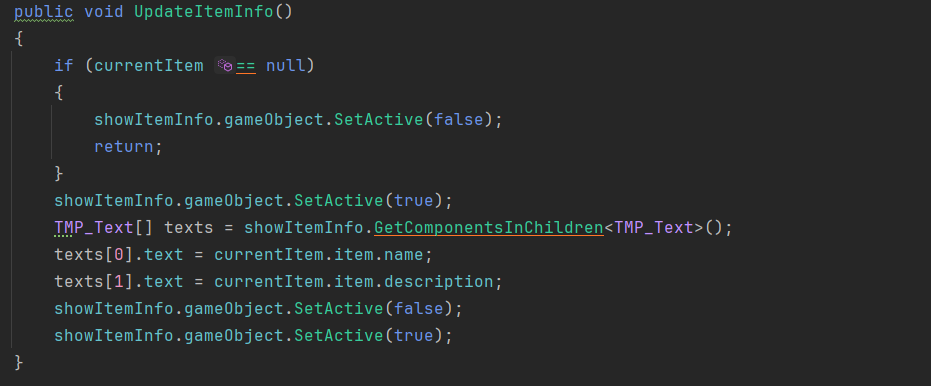
1. 物品信息悬浮显示
   1. 判断鼠标悬浮：通过IPointerEnterHandler和IPointerExitHandler接口来判断鼠标是否悬浮在ItemUse的UI上，并通过isEnter记录是否悬浮以及对BackpackManager中的currentItem赋值。



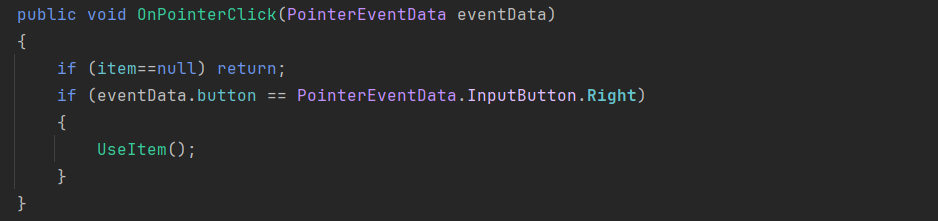
* 1. 更新物品显示面板的位置：使用ScreenPointToLocalPointInRectangle方法将屏幕坐标转为背包画布的局部坐标，再对显示面板的位置赋值，代码如下：



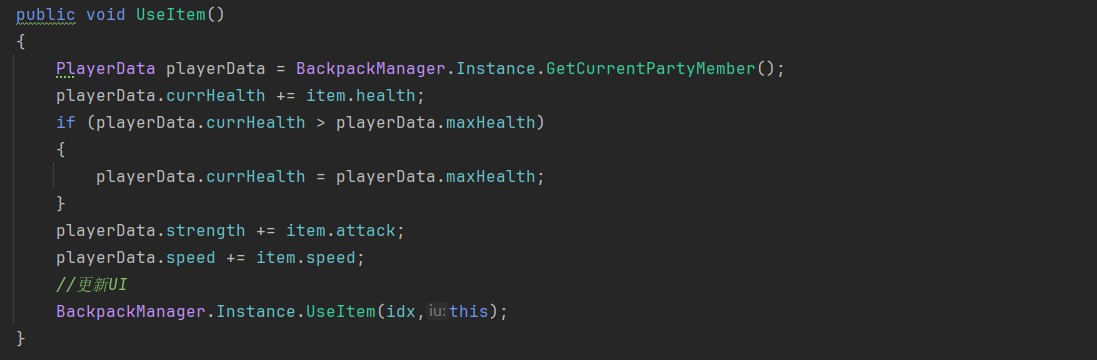
* 1. 悬浮显示物品信息：通过判断BackpackManager中的currentItem是否为空来判断是否显示悬浮面板，若不为空则显示物品的名称和描述，代码如图。由于悬浮面板使用了ContentSizeFitter自动匹配画布大小，因此需要隐藏再显示刷新画布的宽高信息。



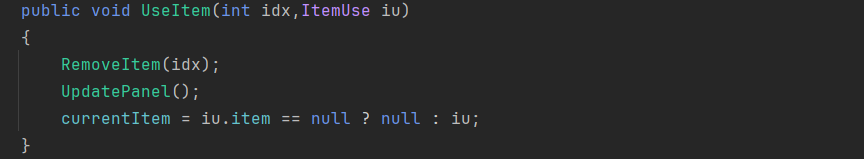
1. 物品使用逻辑
   1. 右键使用物品：通过IPointerClickHandler检测右键点击，若当前的item为空，则直接返回不做任何操作，具体代码如下：

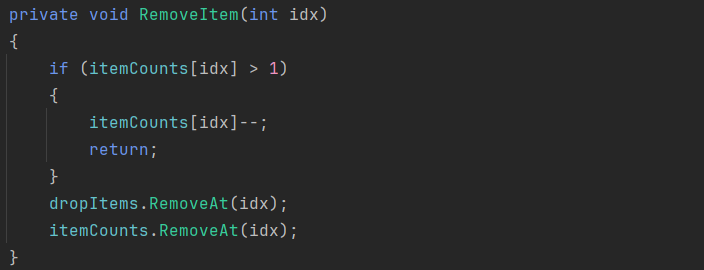


* 1. 数据和UI更新：首先通过单例获取到当前使用物品的PlayerData角色数据，将item的属性信息应用到角色数据上，再调用BackpackManager的UseItem方法通知背包管理器更新当前的背包数据，ItemUse中使用的函数如下：



1. 背包UI更新
   1. 更新成员变量数据：通过ItemUse中UseItem方法传入idx和ItemUse（iu）参数，首先移除当前索引的数据，随后更新当前的UI，并通过判断更新后iu所在格子的ItemUse中的item是否为空来为currentItem赋值。

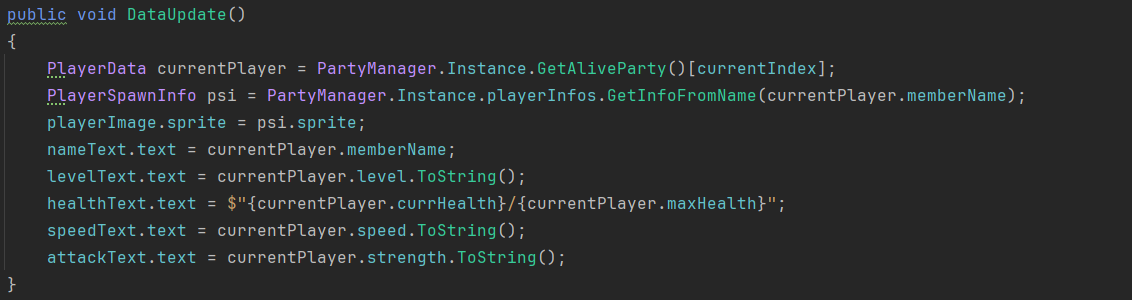




* 1. 更新背包UI：对所有背包格子的ItemUse信息重新赋值，刷新当前UI显示，代码如下：

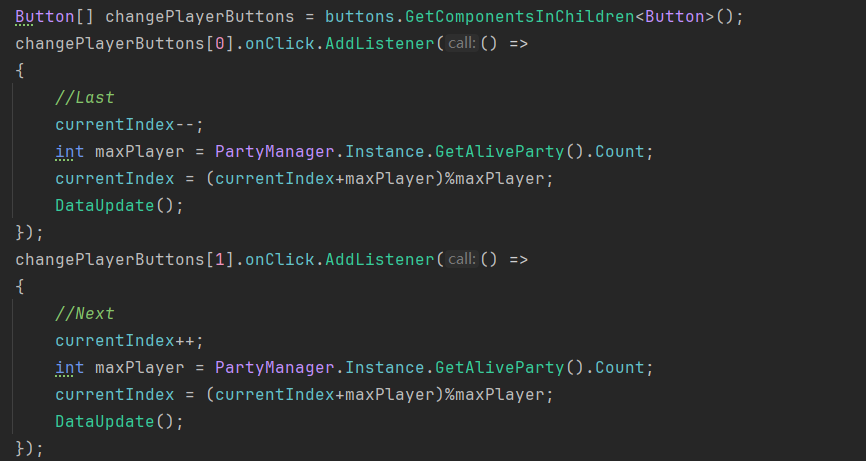


* 1. 更新角色UI：在更新完背包UI之后，会调用DataUpdate对当前角色的数据进行更新，函数的具体代码如下：



1. 角色切换逻辑

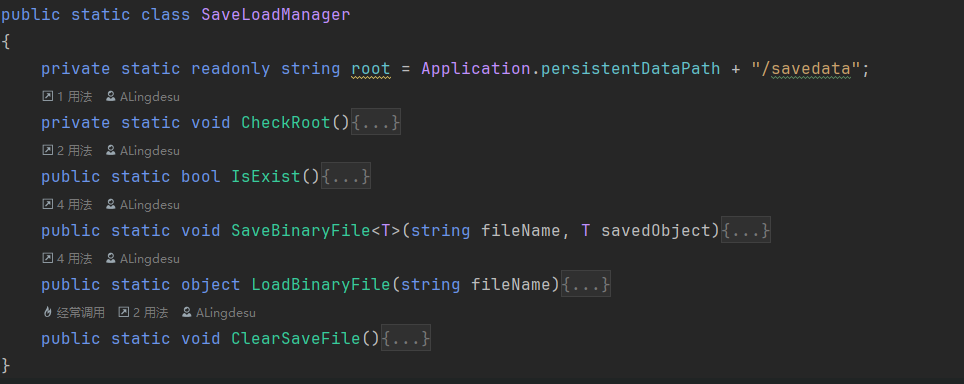
对角色切换的两个按钮添加点击事件，用currentIndex来记录当前角色的索引，再对角色面板的信息进行数据更新。



# 保存系统设计

1. SaveLoadManager静态类设计

创建静态类SaveLoadManager脚本，使用静态类可以方便程序内的所有脚本通过SaveLoadManager.方法 直接访问保存和加载函数，其中包括以下成员：

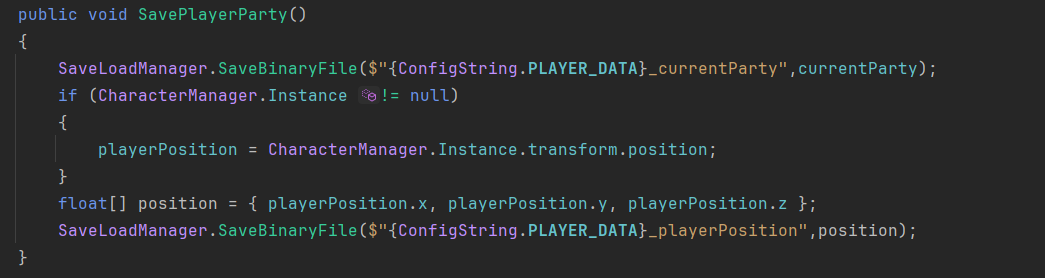


IsExist函数用来判断当前文件夹是否存在、CheckRoot会先检查测文件夹是否存在，若不存在则创建文件夹、ClearSaveFile用于清除当前文件夹保存的数据信息。

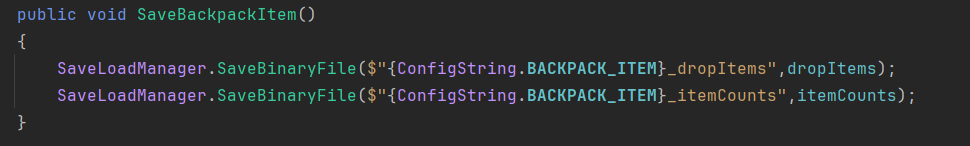
SaveBinaryFile<T>用于保存泛型数据，LoadBinaryFile通过文件名来加载数据并反序列化为object对象，具体代码如下：



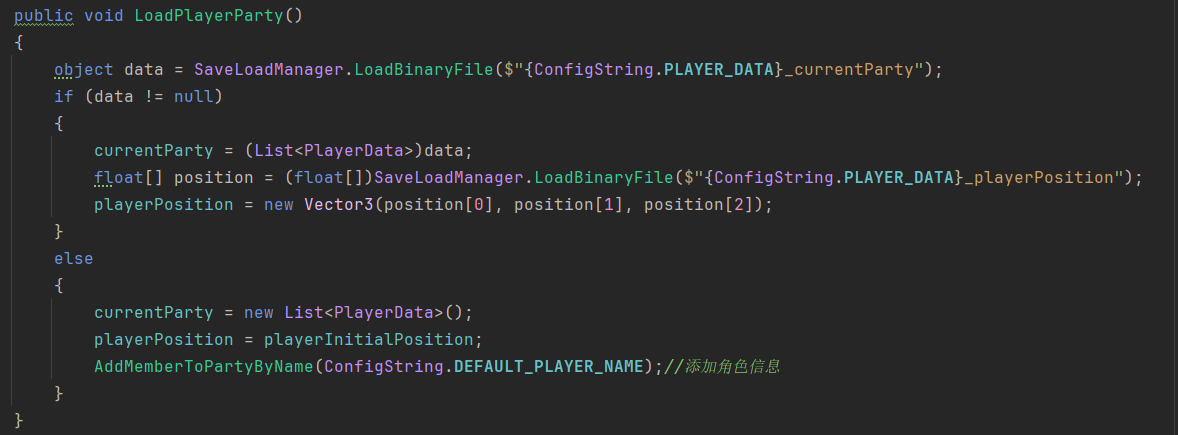
1. 场景内数据的保存和加载
2. 保存数据
   1. 保存角色信息：在保存时，将currentParty中每个PlayerData数据进行序列化保存在本地；获取CharacterManager获取到当前主角色在场景中的位置并保存为float数组（Vector3数据不可序列化），代码如下：



* 1. 保存背包信息：对BackpackManager中的dropItems和itemCounts数据保存，代码如下：



1. 加载数据
   1. 加载角色信息：通过文件路径加载数据，若返回值不为空则说明存在相关存档，则通过强制转换将返回的object对象转为对应的数据类型，再通过赋值的方式完成数据的加载；若返回值为空则进行初始化操作，代码如下：



* 1. 加载背包信息：与加载角色数据一样，代码如下：



# 战斗状态线条指引

1. 贝塞尔曲线
2. 基础概念

贝塞尔曲线(Bézier curve)是计算机图形学中广泛应用的一种参数化曲线，由法国工程师皮埃尔·贝塞尔(Pierre Bézier)在1960年代为汽车设计而开发，该曲线具有以下特性。

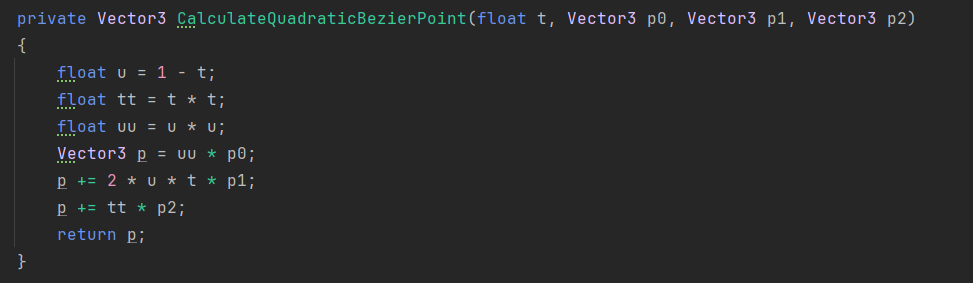
* 参数化曲线：用数学公式描述平滑曲线
* 控制点定义：曲线形状由控制点决定
* 插值特性：曲线通过起点和终点，但不一定通过中间控制点

对于n阶贝塞尔曲线，t时刻的点位置为：

其中为二项式系数，为控制点坐标，为参数。

1. 代码实现

在程序实现中使用简单的2阶贝塞尔曲线，计算3个控制点下t时刻的点坐标代码为：



1. 曲线更新
2. 脚本创建

创建AttackLine的脚本并挂载到名为LineRenderer物体上，该物体挂载LineRenderer组件，用于绘制曲线线条，最后将该物体作为BattleSystem的子物体，便于BattleSystem获取调用。

1. 基础属性

设置AttackLine的基础属性，包括：

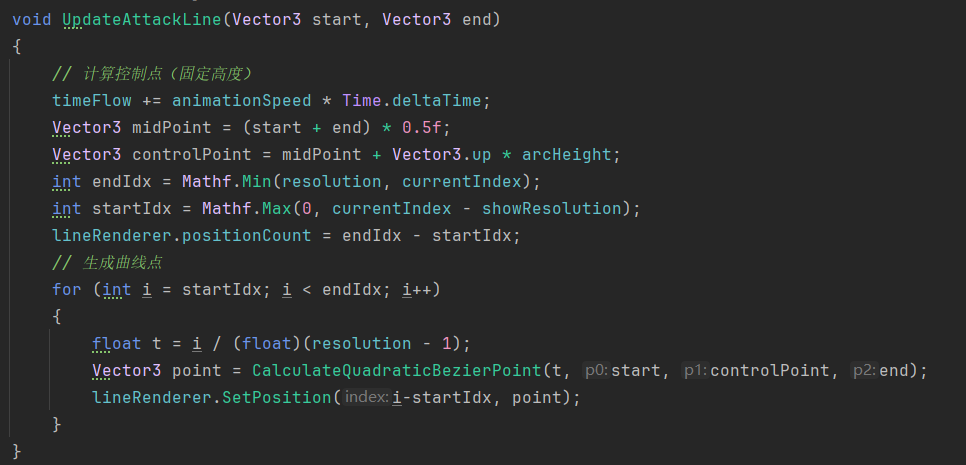
* LineRenderer lineRenderer LineRenderer组件
* Transform startPoint 起点的transform
* Transform endPoint 终点的transform
* float arcHeight 中间控制点的高度
* int resolution LineRenderer点总数量
* int showResolution 显示的点数量
* float animationSpeed 动画速度
* Color startColor 起始颜色
* Color endColor 终止颜色
* int currentIndex 当前的初始点索引
* float timeFlow 当前的时间

属性的初始化如下：



1. 更新策略

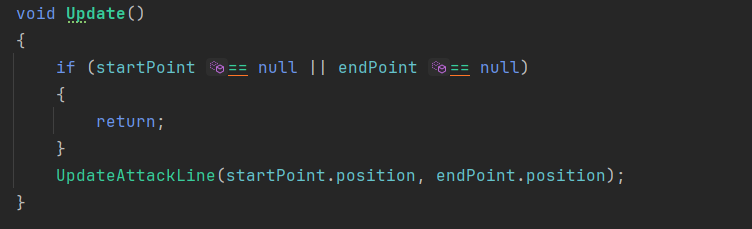
线条更新的代码如下：



首先计算当前的时间和中间控制点的位置，随后计算需要显示的点的开始索引和结束索引，其中currentIndex = (int)timeFlow % (resolution + showResolution)，由于需要在这resolution个点中选取连续的showResolution个点进行显示，因此采用对(resolution + showResolution)取模的操作来获取当前时间的点索引。

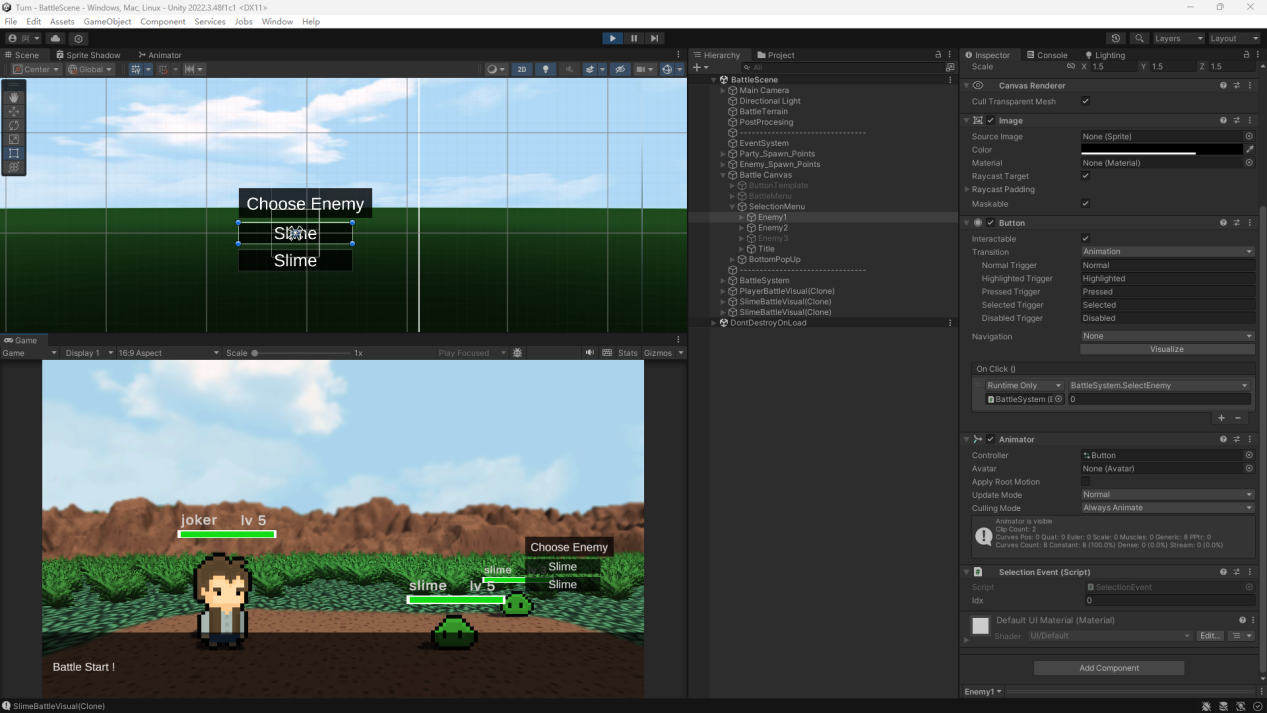
然后通过计算当前时间的开始点索引和结束点索引来确定显示的点范围，最后遍历所有需要显示的点索引来计算每个点的位置并赋值给LineRenderer组件。

在Update方法中，通过判断当前startPoint和endPoint是否为空来决定是否更新曲线的动画：

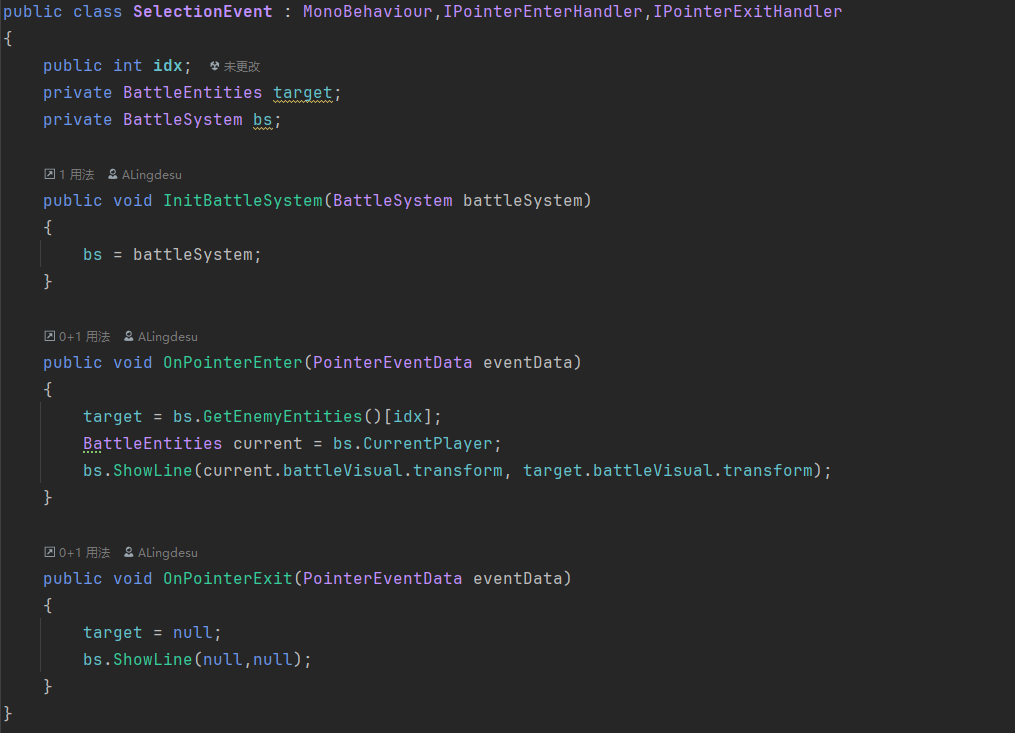


1. 调用线条绘制
2. 按钮的脚本创建

创建SelectionEvent脚本并挂载到敌人选择菜单的每个按钮上：



该脚本继承IPointerEnterHandler,IPointerExitHandler两个接口来管理鼠标进入和离开按钮图标的事件，具体的脚本代码如下：



其中idx为当前按钮代表的敌人索引，target为当前按钮指向的敌人实体，bs为场景中的BattleSystem，用于获取当前操作的角色并调用曲线绘制方法。

1. 调用策略

当鼠标进入按钮时会通过bs战斗系统脚本获取到当前的角色和攻击的目标实体，再传入相关的Transform组件给到BattleSystem调用绘制线条的函数；当鼠标离开时则传入null参数。

BattleSystem将接收到的参数再传给AttackLine的SetTransform方法给AttackLine的起点和终点组件进行赋值，从而实现战斗状态的曲线指示动画。

