游戏设计文档

基本信息

- 项目名称：Blue Protocol

开发环境

- Unity 2020.3.45(及以前版本，后面的版本可能有问题) 3D URP
- 版本控制 Git，仓库地址。

其它说明

素材网站

动画

- https://www.mixamo.com/#/ 单纯的动画网站 Mixamo，能找到多数的一般动作；
- https://assetstore.unity.com/ 资产商店，每周开发者能嫖一个付费资产；

音效/配乐

- https://www.aigei.com/ 上面音效配乐为主，也有一些特效 UI 等素材；

模型

- https://www.aplaybox.com/ 模之屋的模型以二次元为主，但是使用需要用 Blender 的 cats 插件将 pmx 转成 fbx，具体看这篇博客；
- https://gamedev3d.com/forum-92-1.html 游研堂除了模型还有动作，不过感觉比较少，而且好像多数要付费；
- https://texture.ninja/ 一个纹理网站。

设计文档

UI 设计

开始画面

点击 Start 开始游戏，Exit 退出游戏。

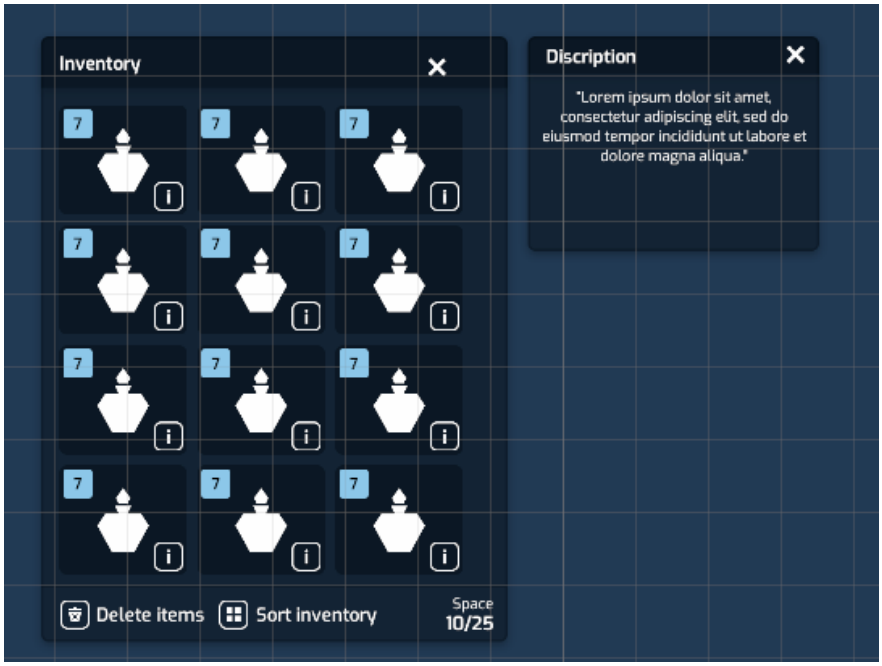
按键说明

鼠标控制镜头方向，WASD 控制移动，鼠标左键攻击，空格翻滚，F 对话交互，B 打开背包，点击物品使用或查看详细描述。

HUD

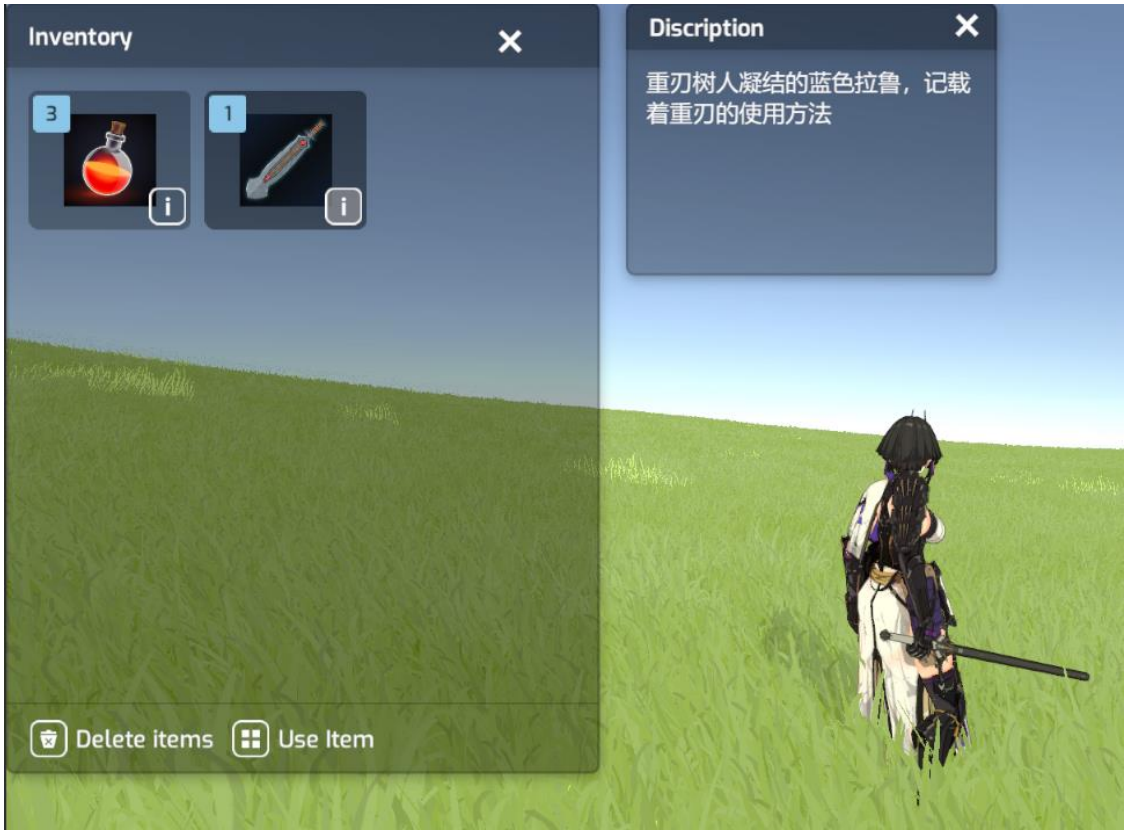
主要是敌人和玩家的学校，靠近敌人时显示敌人血条，敌人死亡后敌人的血条消失；玩家的血条一直存在。

背包

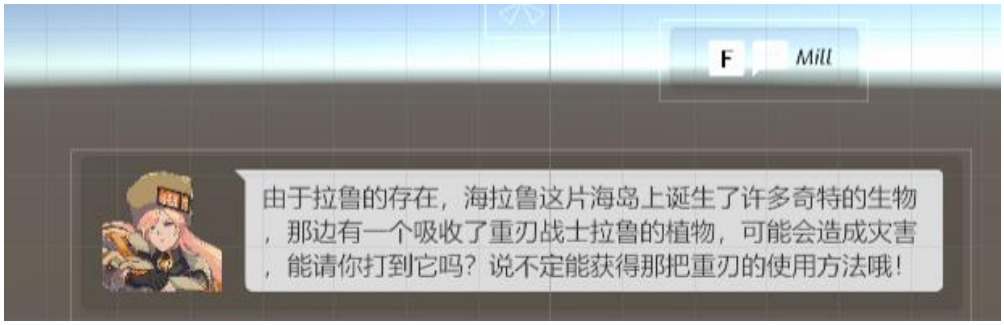


按 B 打开背包，再次按 B 关闭背包，点击图标使用物品，图标左上角显示物品数量，点击右下角显示详细描述；点击下方 Delete items 删除物品，点击 Sort inventory 进行排序，Space 表示当前空间。

当前设置两种物品，血瓶属于 Portion 类别的物品，使用后将回复全部生命值；大剑属于 Weapon 类别的物品，使用后角色掌握大剑的使用，并且可以切换武器。



对话



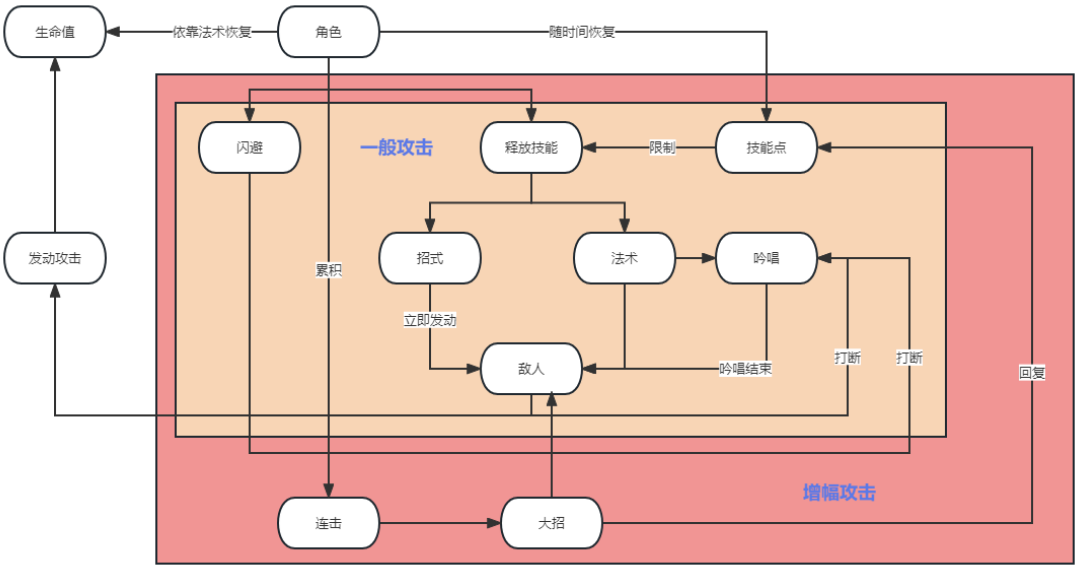
靠近 NPC 显示对话按键提示，按键触发后显示对话框，继续按键显示下一条对话。

战斗设计

战斗机制

概述

战斗中玩家的行为可以用下图概括：



攻击方式

攻击分为增幅攻击和一般攻击。

一般攻击根据场上角色有不同的攻击方式，目前指定战士和法师两种，攻击方式也分为两种，招式和法术，这两种攻击方式战士和法师都可以使用，但是战士的招式攻击占比大，法师的法术攻击占比大。角色使用这两种攻击时需要消耗技能点，根据技能点花费的多少，技能具有不同的伤害幅度，而且招式可以立即发动，而多数法术则需要吟唱，被打断则法术释放失败。敌人的攻击和角色主动闪避都会打断法术的吟唱。

增幅攻击是当角色的连击数累计到一定数值以后，角色此时可以释放特殊的攻击。

目前角色具备剑、大剑两种攻击方式，各有几种攻击招式，攻击招式可以任意衔接。

一般攻击状态机

- 如果 1s 内没有再次输入 attack，且 Attack 播放完成，转到默认状态；
- 如果处于默认状态，按下攻击，转到 Attack1；
- 如果 1s 内按下攻击且当前状态为 Attack1，且播放大于 0.3，转到 Attack2；
- 如果 1s 内按下攻击且当前状态为 Attack2，且播放大于 0.3，转到 Attack3；

数值设计

游戏中战斗的数值方面分为生命值，攻击力，防御力，还有技能点，连击数 5 种数值，其中除技能点和连击数是角色特有外，前四种属性敌人和角色是共有的。

- 生命值是一个单位在战斗中存活的依据，减小到 0 后单位死亡；
- 攻击力是单位攻击后造成伤害的依据；
- 防御力能使单位对受到的伤害进行衰减；
- 技能点限制技能释放；
- 连击数随攻击命中累计；

实际伤害=max(0,攻击力*技能倍率/(防御力/攻击力))

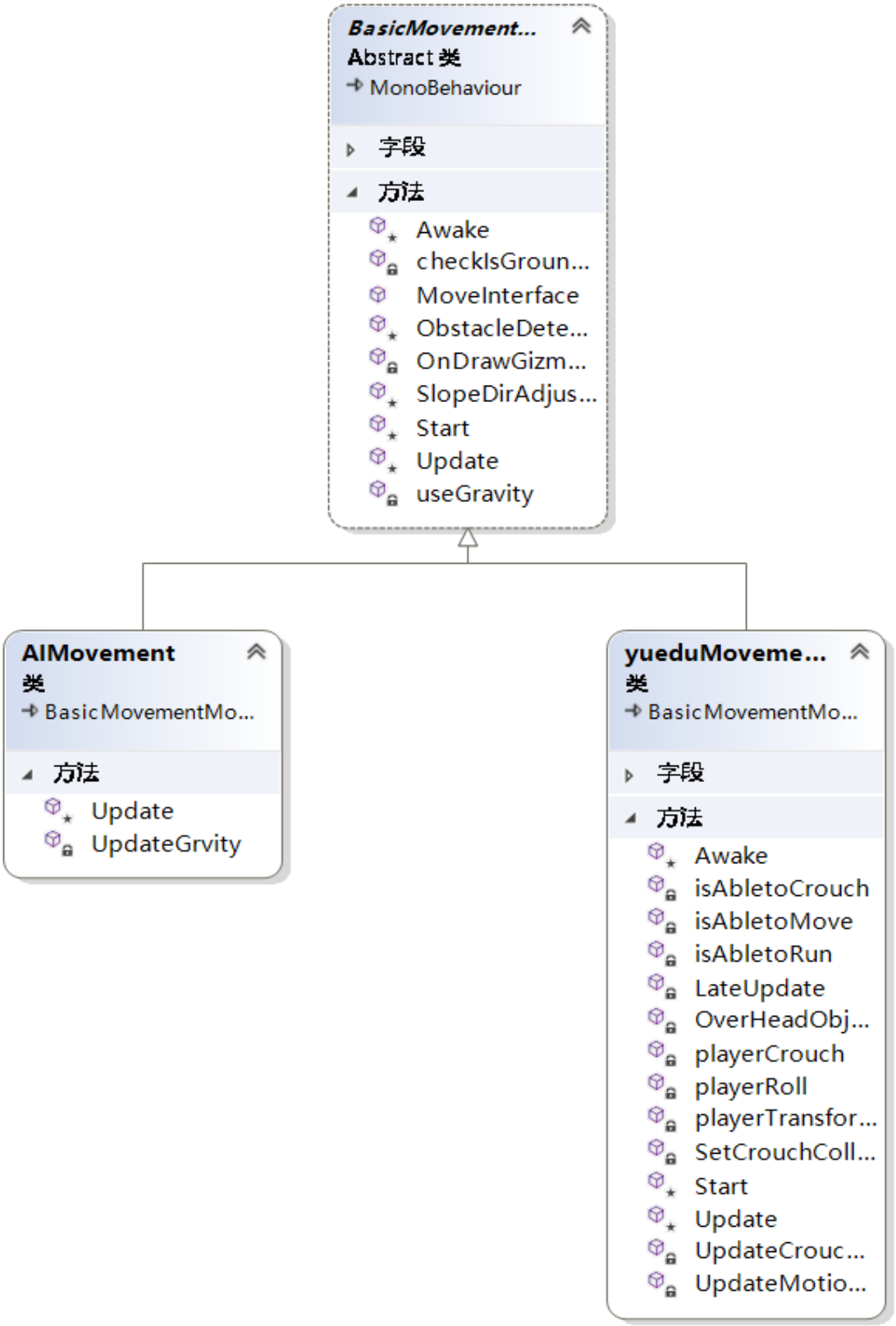
关卡设计

游戏中包含一个原型关卡，主角从 NPC 处接取任务后，执行任务内容，也就是打败敌人，之后获得敌人掉落的拉鲁，然后角色与拉鲁互动获得新的武器，并回到 NPC 处提交任务，获得奖励。之后能选择利用获得的道具继续和敌人进行战斗。

系统框架设计（Gameplay）

游戏系统主要有移动系统，战斗系统，AI 系统，属性系统，UI 系统。

移动系统框架


BasicMovement...
Abstract 类
MonoBehaviour
字段
方法
Awake
checkIsGroun...
MoveInterface
ObstacleDete...
OnDrawGizm...
SlopeDirAdjus...
Start
Update
useGravity
AIMovement
类
BasicMovementMo...
方法
Update
UpdateGrvity
yueduMoveme...
类
BasicMovementMo...
字段
方法
Awake
isAbletoCrouch
isAbletoMove
isAbletoRun
LateUpdate
OverHeadObj...
playerCrouch
playerRoll
playerTransfor...
SetCrouchColl...
Start
Update
UpdateCrouc...
UpdateMotio...

游戏中几乎所有角色都需要移动，因此有必要抽取基类作为抽象类实现共有的功能。角色均使用 character controller 组件实现移动，character controller 的 Move 方法不提供重力模拟，所以基类要提供的功能是：地面检测，障碍物检测，斜面方向调整，重力模拟，以及必要组件的引用。注意我们不在基类中计算移动方向，因为玩家和其它角色的移动方向不同，玩家需要根据摄像机来更正方向，而其它角色则由 AI 控制，因此我们这里仅给出一个公开的移动方法，该方法接受移动方向、移动速度、是否启用重力为参数，根据这些值调用 character controller 的 Move 实现移动。

```
#region 内部方法
/// <summary> 施加重力
1 个引用
private void useGravity()...

/// <summary> 射线检测是否在地面上
1 个引用
private void checkIsGrounded()...

/// <summary> 修正移动dir
2 个引用
protected Vector3 SlopeDirAdjustment(Vector3 dir)...

/// <summary> 检测障碍物
1 个引用
protected bool ObstacleDetection(Vector3 dir)...

#endregion
```

其中地面检测使用 CheckSphere 方法配合 LayerMask 完成；障碍物检测根据移动方向进行 Raycast 检测是否前方存在障碍物；重力模拟首先检测是否在地面上，不在地面上则添加 Y 方向上的速度完成重力模拟。而由于 Move 方法不能模拟重力，当角色在斜面上行动的时候需要调整移动方向平行于斜面，所以要根据斜面法线将当前的移动方向投影到斜面上去。

玩家的移动

除了基类方法外，玩家还具备下蹲和奔跑，翻滚功能，为保险起见我们先给出三种判断方法分别判断是否能移动、是否能下蹲、是否能奔跑，翻滚没有条件。移动条件为，玩家在地面上，并且当前状态机播放的动画 Tag 也为移动标签；下蹲条件为，当前角色没有在奔跑，并且动画状态机并没有在播放下蹲的过渡动画，蹲起的

时候检测角色头顶是否存在障碍物；奔跑条件为，当前能够移动，并且目标移动方向和当前角色朝向基本一致，也就是说奔跑前需要等待角色转身。

```
/// <summary> 是否能移动
2 个引用
private bool isAbletoMove()...

/// <summary> 是否能下蹲
1 个引用
private bool isAbletoCrouch()...

/// <summary> 是否能奔跑
1 个引用
private bool isAbletoRun()...
```

首先是移动方向计算 ，摄像机采用自实现的脚本完成一个带碰撞的第三人称相机，而相机方向更改人物移动主要是 Yaw 偏航角，也就是相机目前旋转的 Y 值。所以我们先获取输入，用反正切函数计算实际前进方向和当前角色前方朝向的夹角，然后加上相机的 Y 轴旋转值，他们都是往右为正，往左为负，这样就更正了移动方向。然后根据输入用插值方法设置当前的移动速度，根据目标方向旋转角色，调用 Move 方法移动角色。

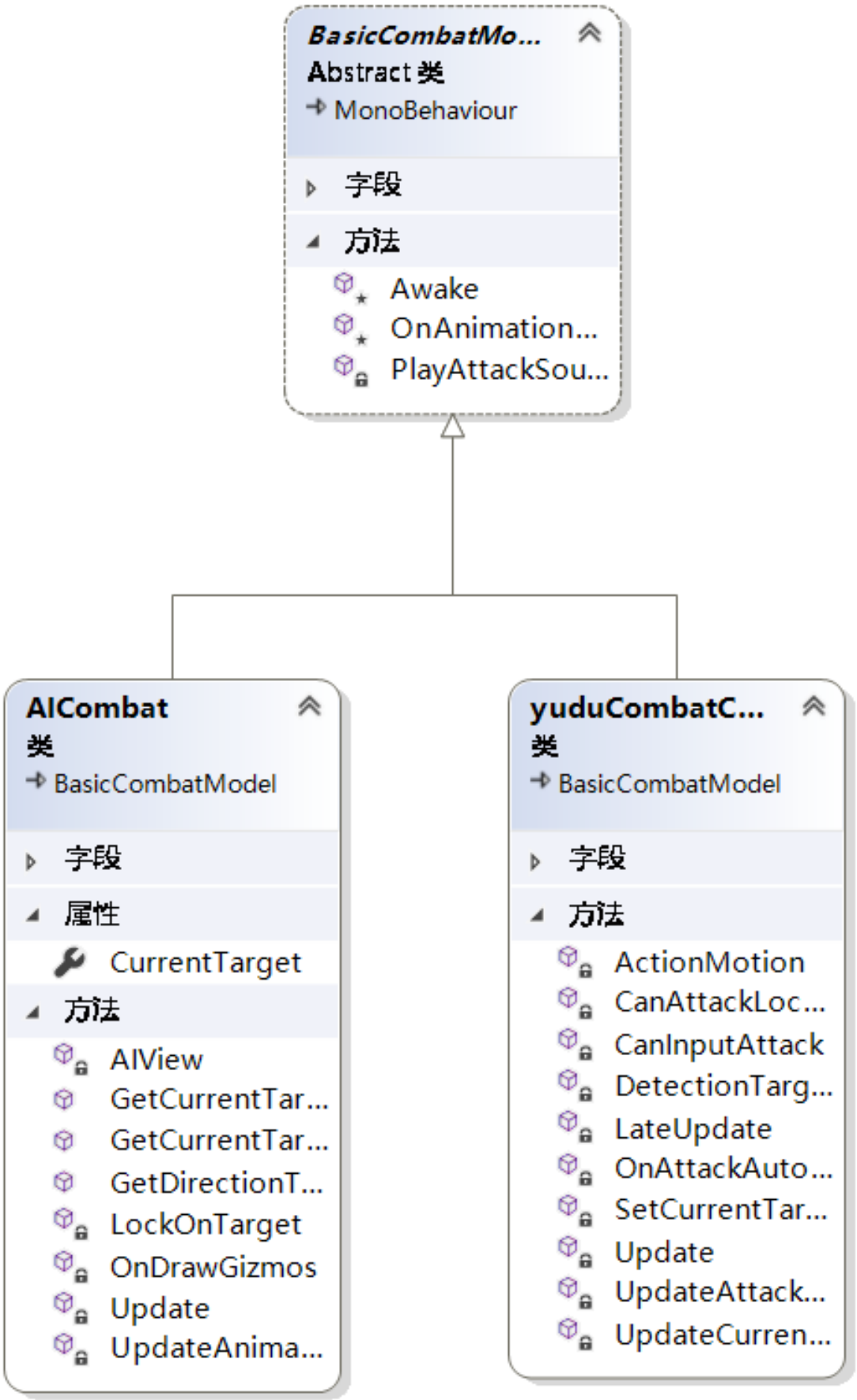
然后是下蹲的实现，如果下蹲条件方法为真，那么就用 bool 值更换角色的下蹲状态，注意还要设定新的碰撞体高度和半径。翻滚采用 Trigger 完成，由于动画不具备 Root Motion，所以这里的解决方法是采用 Animation Curves 为翻滚动画设置一个可获取的状态机参数作为翻滚时的移动速度，调用基类的移动接口完成翻滚的移动。

敌人的移动

敌人的移动相对简单，由于不需要任何输入，所以采用基类方法就能完成，主要在 AI 的控制上，后面详细介绍 AI 的移动控制。

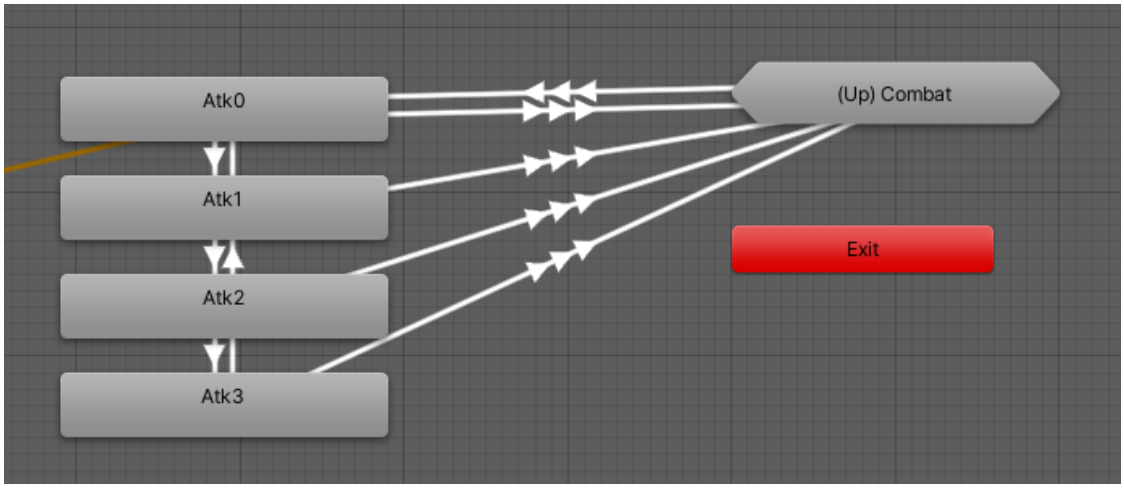
战斗系统框架

战斗体现为攻击，这种行为是玩家和敌人共有的，所以也需要抽取基类完成。基类需要实现的功能是攻击检测，玩家和敌人的攻击方式不同，需要通过继承实现。攻击检测方案为，在角色前方设置一个球形检测区域，如果攻击时敌人其中就判断为受到攻击了，此时根据碰撞信息调用敌人方法让敌人接受伤害，并使用资源类播放音效。

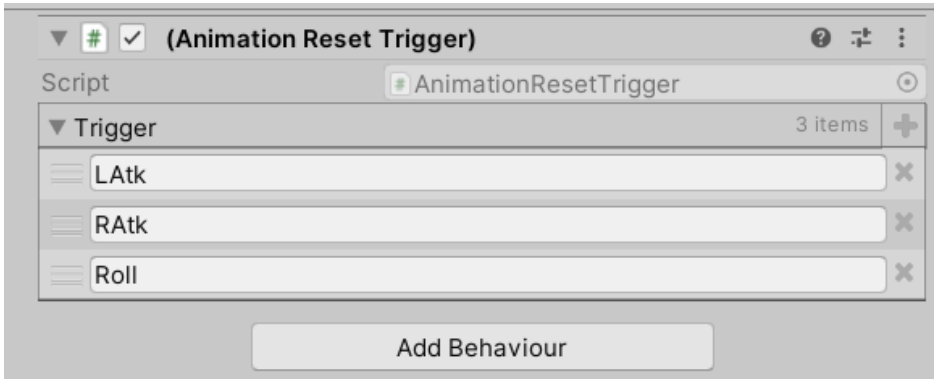

BasicCombatMo...
Abstract 类
MonoBehaviour
字段
方法
Awake
OnAnimation...
PlayAttackSou...
AICombat
类
BasicCombatModel
字段
属性
CurrentTarget
方法
AIView
GetCurrentTar...
GetCurrentTar...
GetDirectionT...
LockOnTarget
OnDrawGizmos
Update
UpdateAnima...
yuduCombatC...
类
BasicCombatModel
字段
方法
ActionMotion
CanAttackLoc...
CanInputAttack
DetectionTarg...
LateUpdate
OnAttackAuto...
SetCurrentTar...
Update
UpdateAttack...
UpdateCurren...

玩家连招

玩家具有两种装备，Sword 有四种攻击方式，Great Sword 有三种攻击方式，招式之间可以相互衔接。连招使用 Trigger 触发，并且使用 Reset Trigger 取消重复触发，保证每个动画中攻击按键仅一次有效，防止多次误触。



为保证输入有缓冲时间，让玩家能够在角色攻击动画没有播放到给定时间之前不能切换状态，攻击动画未完全结束之后按键随时切换状态，需要通过判断播放时间完成。那么 ResetTrigger 实际上有两个地方需要使用，一个是 OnStateEnter 中需要执行，保证触发器只会影响当前动画；另一个是判断能否更新攻击时，如果判断不能返回，那么一并要消除此时没有触发的触发器，否则即使返回了 False，实际上状态机中的 Trigger 并不是 False，因为状态机某种 Bug，如果多次触发 Trigger，那么 OnStateEnter 中的 Reset 似乎不会起效。



有时候重启能解决该问题，不过保险起见还是在返回 False 是重置 Trigger。

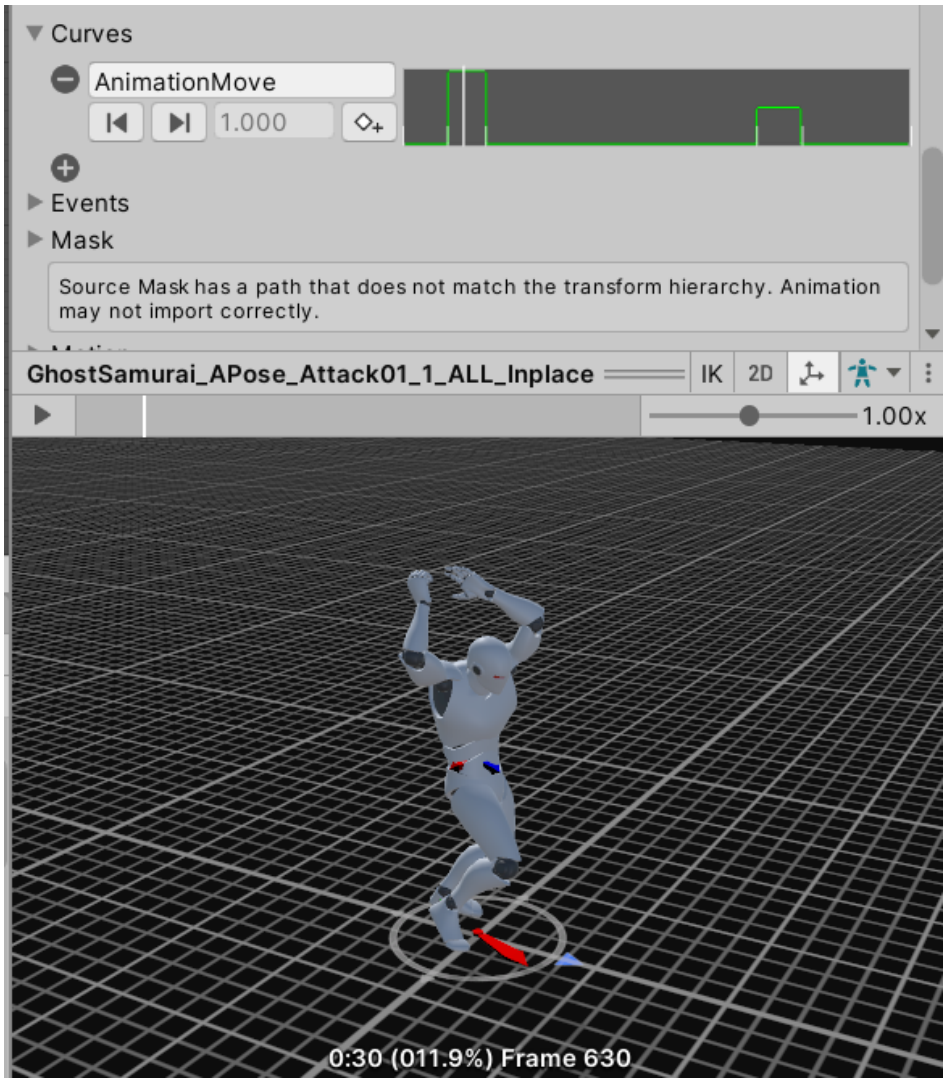
攻击矫正

另外为保证攻击手感，需要给角色的近距离攻击添加一个锁定目标，当角色靠近敌人攻击时需要让角色旋转朝向敌人，从而自动矫正攻击方向，否则玩家会经常因为攻击方向不正确而导致无效攻击。

```
#region 攻击自动索敌
/// <summary> 旋转到锁定对象
1 个引用
private void OnAttackAutoLockOn()...
/// <summary> 判断攻击状态是否允许自动锁定敌人
1 个引用
private bool CanAttackLockOn()...
/// <summary> 检测前方敌人
1 个引用
private void DetectionTarget()...
/// <summary> 设置当前锁定目标
1 个引用
private void SetCurrentTarget(Transform target)...
/// <summary> 更新锁定状态 移动时不能自动锁定
1 个引用
private void UpdateCurrentTarget()...
#endregion
```

攻击移动

攻击时角色的动画应该是有位移的，不过由于是 Inplace 动画，所以需要使用 Animation Curves 添加位移，调用移动基类方法完成攻击动画的移动。




```
/// <summary>
/// 控制攻击时的移动
/// </summary>
1 个引用
private void ActionMotion()
{
    if (animator.CheckAnimationTag("Attack") || animator.CheckAnimationTag("GSAttack"))
    {
        MovementBase.MoveInterface(transform.forward, animator.GetFloat(animationMoveID) * animationRootAttackScale, true);
    }
}
```

武器切换

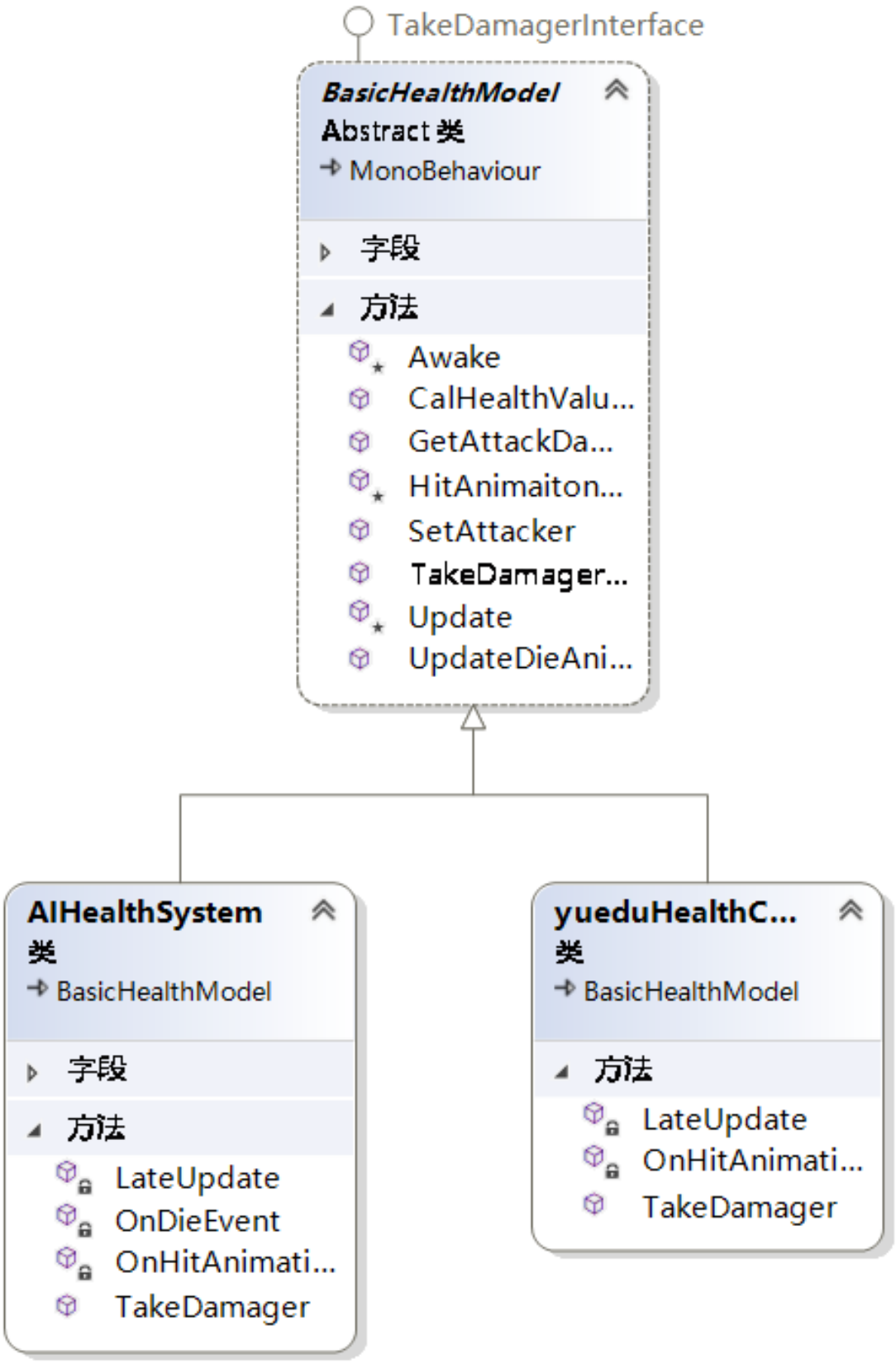
角色在打败敌人后能获得第二种武器，此时需要在两种武器之间进行切换。这里的解决方案是，将这些武器分别绑定在角色的手部骨骼和胸部骨骼后方，正常情况下如果装备 Sword，那么启用手上的 Sword，禁用背上的 Sword；启用背上的 Great

Sword，禁用手上的 Great Sword。装备 Great Sword 时同理。这里我选择用一个单独的装备类来管理角色的装备状态，他获取游戏对象的引用，然后分别在初始化、拾取大剑、切换武器时更改启用情况。

属性系统框架

角色拥有自己的属性，包括生命值，攻击力等信息，应该使用一个抽象类作为基类提供这些属性的更改方法，因此之前的接受伤害方法也会在这里实现。基类应该实现的主要功能是：

- 提供获取攻击力的外部方法，提供接受伤害的外部方法，这样可以通过动画事件的回调函数获取受击对象的属性系统类，该方法中还要设置当前的攻击者对象，调用接受伤害的外部方法获得攻击力，并实现受击伤害的结算。注意此时还要播放受击动画。
- 提供死亡事件方法，可重写，这样在子类中可以根据需要定制死亡事件，例如敌人死亡会掉落拉鲁，角色死亡会弹出结算 UI；
- 提供必要组件的引用，尤其是 UI 管理器，UI 管理器是 UI 系统的核心，后面会详细说明。


TakeDamagerInterface
BasicHealthModel
Abstract 类
MonoBehaviour
字段
方法
Awake
CalHealthValu...
GetAttackDa...
HitAnimaiton...
SetAttacker
TakeDamager...
Update
UpdateDieAni...
AIHealthSystem
类
BasicHealthModel
字段
方法
LateUpdate
OnDieEvent
OnHitAnimati...
TakeDamager
yueduHealthC...
类
BasicHealthModel
方法
LateUpdate
OnHitAnimati...
TakeDamager

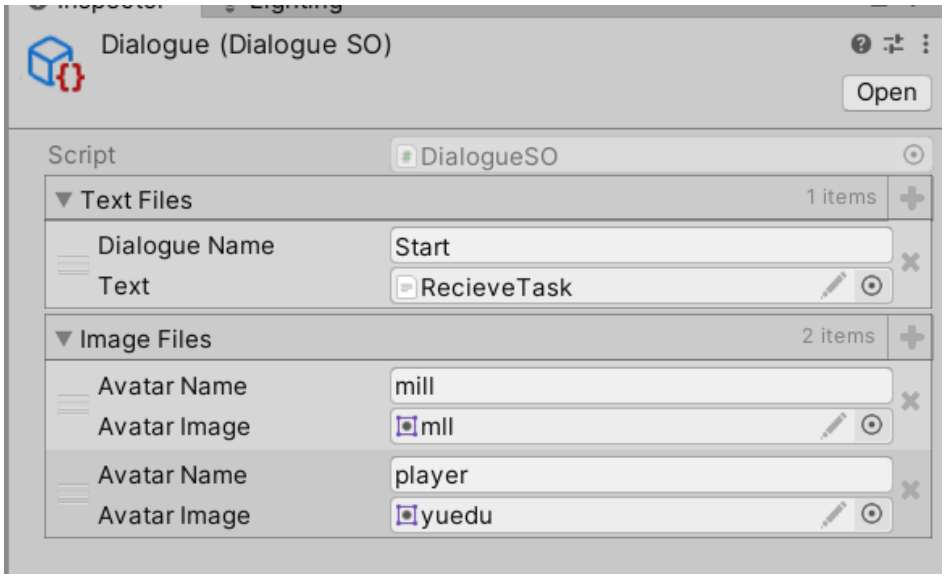
此外还有一些封装属性的其它小方法。下面我们需要给玩家和敌人分别继承来实现各自的属性系统，主要是重写死亡事件和伤害接受方法。此外为了优化战斗过程，还要添加一个受击旋转方法，让敌人和角色受击后旋转到面朝攻击者的方向，这样能让玩家方便调整角色姿态，因为攻击的同时无法移动。

UI 系统框架

所有需要使用的 UI 都通过 Canvas 下的 UIManagement 管理，该子对象用一个管理类获得所有 UI 的 Panel 的引用，除必要的内部方法外，提供封装好的外部方法来供所有其它类调用，也就是说所有会更改 UI 的对象动作都需要获得这个 UIManagement 的引用，这样就方便了 UI 的管理。我们接下来在之前的内容上进一步添加 UI 的功能。

对话系统

根据剧情推进，角色和 NPC 的互动会发生变换，也就是需要对话框使用不同的角色头像，不同的文本内容，可以采用编写一个对话资源 Scripts Objects 来完成，资源用枚举类型根据具体任务分类存储文本配置，以及角色对话头像。具体的对话文本用 txt 存储，头像为 Sprite 类型，加载到 SO 上供对话系统读取。



解析 Text 之后将文本按行存储在一个字符串 List 中，随按键触发更新到对话框的文本组件中，并根据文本中的标注选择获得对话头像。每个 NPC 的对话都使用该系统完成对话。

另外为了让字符能够逐个显示，采用协程的方式完成输出延迟，bool 标识当前对话是否完成，只有完成对话才能进入下一个协程显示下一段对话的字符。

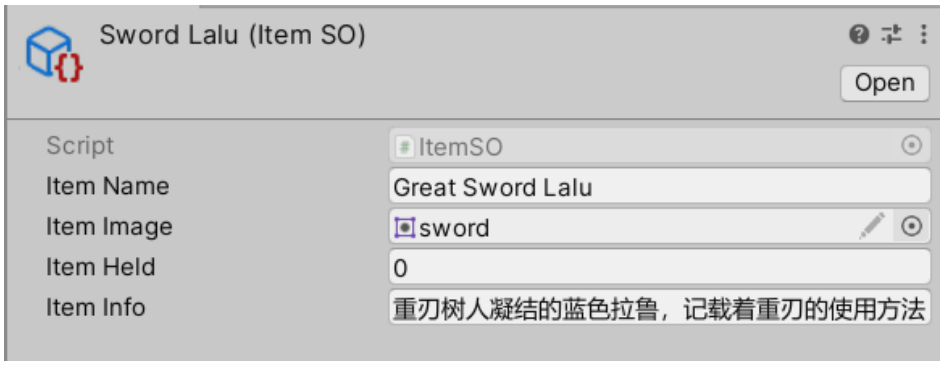
```
private IEnumerator SetTextUI()...

#endregion


#region 外部方法
1 个引用
public void UpdateDialogueText()
{
    if(index == textList.Count)
    {
        gameObject.SetActive(false);
        index = 0;
        return;
    }
    if(textList.Count == 0)
    {
        return;
    }
    if (textFinished == false) return;
    StartCoroutine(SetTextUI());
}
```

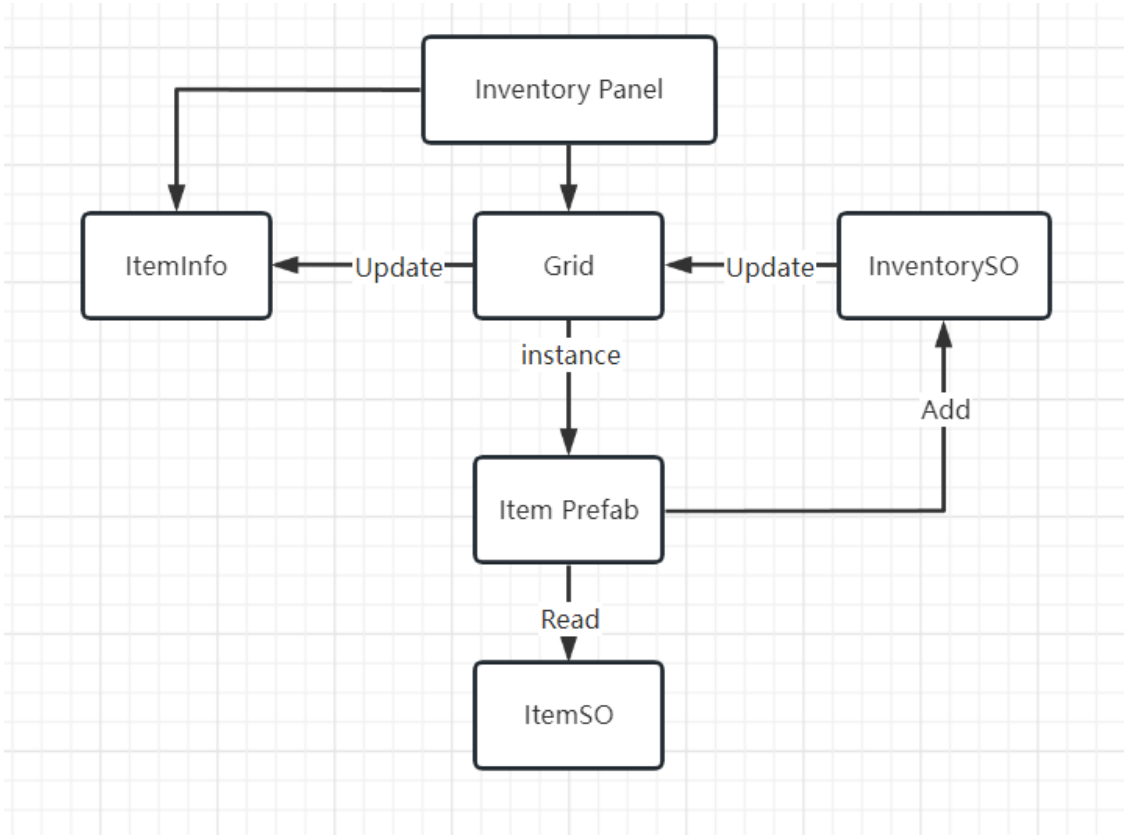
背包系统

角色拾取装备或物品后后在背包中可以选择查看、更换武器，每个物品作为背包的 Item 资产，该资产也是 Scripts Objects，包含物品对应的图片和描述文本，目前数据持久化同样使用 SO 完成，因此还包括物品的数量，在编辑器中此方法是可行的。



此外，还需要为背包设计一个 SO 存储所有的 Item：

此时 UI 可以通过读取 Inventory 中 Item 的信息更新背包的内容。不过要注意，使用 Scripts Objects 进行数据持久化的方法只能在编辑器中作为功能测试使用，更普遍的做法是使用 Json 或者 Database 存储物品的 ID 和信息。



获取物品时通过实例化将图片添加到 Grid 中，物品信息通过读取 SO 配置得到，并首先添加到背包的 SO 中，再进一步添加到 Grid 中显示出来，这样的目的是方便更改背包内容物时刷新 Grid 显示。

任务系统框架

RPG 游戏中任务也是很重要的一部分，之前的对话系统也算任务系统的一个子系统，只不过为了规范把它单独作为一个系统提供一些外部方法供任务系统调用。任务目前仅有主线任务，因此主任务 id 只有一个，它标识了任务的总体；子 id 是任务的细节，例如角色执行一个动作，因此当前任务的子 id 就是三个，接取任务，击败敌人，完成任务。

任务配置采用 Excel 转 Json 完成，我们首先配置该主线任务的配置表如下：

TaskId	Target	DialogueName	Award
0	RecieveTask	RecieveTask	null
1	DefeatEnemy	null	null
2	FinishTask	FinishTask	{"HealthPortion":1}

读取配置信息后可以通过设置整形变量 CurrentId 表示当前的任务进度，当角色还没有接取任务时 Id 为 0，此时对话框应该根据配表加载 RecieveTask 的文本，此时没有奖励；当角色接取后，Id 变为 1，角色需要打败敌人；打败后 Id 变为 3，就可以回去找 NPC 交复任务了，并且最后会修改背包的数据，添加三个血瓶。

美术风格

使用卡通渲染风格，管线为 Universial RP，其中场景使用资产自带的 Shader，人物使用 UnityURPToonLitShaderExample 的开源 Shader。

音效设计

包括背景音乐，武器挥动音效，受击音效。使用 Scripts Objects 创建音效分类清单管理音效 Clip，背景音乐默认播放不做管理，音效清单有 Sword，Great Sword，Hit 三种类型的音效。

音效清单获取音效后，初始化方法中用字典存储文件和对应音效名方便查找，给出 GetClip 方法根据类型随机返回该类别中的某个音效。

```
public AudioClip GetClipAssets(SoundAssetsType soundAssetsType)
{
    switch (soundAssetsType)
    {
        case SoundAssetsType.hit:
            return assetsDictionary["Hit"][Random.Range(0, assetsDictionary["Hit"].Length)];
        case SoundAssetsType.swordWave:
            return assetsDictionary["SwordWave"][Random.Range(0, assetsDictionary["SwordWave"].Length)];
        case SoundAssetsType.gSwordWave:
            return assetsDictionary["GSwordWave"][Random.Range(0, assetsDictionary["GSwordWave"].Length)];
        default:
            Debug.Log("没找到");
            return null;
    }
}
```

使用 GameAssets 类继承实现单例，Awake 调用初始化处理文件字典索引，并给出 PlaySoundEffect 方法设置 AudioSource 的 Clip 并播放该 Clip。