💛

角色系统

重点部分：

1. 玩家状态逻辑：玩家移动、动画切换、模型更改
2. 玩家血量/饱食度计算及同步UI显示
3. 玩家数据和模型的保存与恢复

核心脚本：

1. Player_Controller.cs：对外提供玩家模型、数据及一些方法的访问，起到一个承接转发的作用
2. Player_Model.cs
3. PlayerStateBase.cs：玩家状态基类, 抽象出所有玩家状态所需要的共同字段/函数
    a. Player_Attack.cs
    b. Player_Dead.cs
    c. Player_Hurt.cs
    d. Player_Idle.cs
    e. Player_Mode.cs
4. PlayerData.cs
5. PlayerConfig.cs

玩家模型 PlayerModel.cs

玩家模型文件分析：

1. Group：不清楚用处
2. Mesh：玩家身体部位Skinned Mesh Renderer，与普通Mesh Renderer还是有区别的
3. Root_M：玩家身体部位，例如武器挂载位置就是在右手处（Wrist_R）

玩家模型脚本初始化

1. 初始化武器挂载位置：玩家模型右手处（Wrist_R）
2. 初始化模型在动画执行过程中触发的一些事件（委托/Action）例如移动时发出脚步声、攻击开始、停止攻击、攻击结束、玩家受伤、玩家死亡，执行方法方式为 `Action?.Invoke()`

玩家状态 PlayerStateBase.cs 及其子类

初始化方法：

1. 传入Player_Controller，目的是能在状态处理的时候快速找到玩家的一些数据和调用其他类的方法

玩家待机状态 PlayerIdle：

1. 进入状态：循环播放待机动画

2. 状态中：持续检测玩家是否按下水平/垂直方向按键
3. 退出状态：如果按下按键则切换到<玩家移动状态>状态退出待机

玩家移动状态 PlayerMove：

1. 初始化：需要传入`CharacterController`，后续移动角色位置时需要用到`CharacterController.Move`
2. 进入状态：播放移动动画
3. 状态中：
   a. 检查玩家是否按下水平/垂直方向按键，如果没有持续按下，则切换到<玩家待机状态>
   b. 通过Input得到水平/垂直方向按键值计算玩家朝向并使用插值计算提升转向流畅度

```
// 2. 查看是否进行朝向计算
Vector3 inputDir = new Vector3(h, 0, v);
Quaternion targetQua = Quaternion.LookRotation(inputDir);
player.playerTransform.localRotation = Quaternion.Lerp(
    player.playerTransform.localRotation,
    targetQua,
    Time.deltaTime * player.rotateSpeed
);
```

   c. 检查移动是否超过边界
   d. 调用`CharacterController.Move`移动角色
4. 退出状态：无

玩家攻击状态 PlayerAttack：

1. 进入状态：确定攻击方向；播放攻击动画
2. 状态中：持续更新攻击方向，使用`Quaternion.Slerp`随时间平滑进行旋转
3. 退出状态：调用Player_Controller中的OnStopHit方法<停止攻击>

玩家受伤状态 PlayerHurt：

1. 进入状态：播放受伤动画
2. “状态中”和”退出状态”：无

玩家死亡状态 PlayerDead

1. 进入状态：关闭玩家身上的Collider防止播放死亡动画时被攻击进入受伤状态；播放死亡动画

玩家控制器 Player_Controller.cs

初始化方法

1. 读取角色配置和存档数据
2. 初始化角色模型相关的事件（实质是一些Action），详情见 (Player_Model.cs)<玩家模型脚本初始化>
3. 初始化玩家状态机并将玩家状态设置为待机状态
4. 初始化角色位置相关数据
5. 通过事件管理器触发生命值/饱食度UI更新

以下为计算/更新玩家属性的方法

计算角色并更新当前饱食度（Update）

1. 如果当前饱食度>0时根据时间和角色饱食度消耗速度配置再计算新的饱食度结果并触发<角色饱食度UI更新>
2. 如果当前饱食度为0（饱食度为0时开始消耗生命）
    a. 如果当前角色hp>0，根据时间和角色生命值消耗速度配置再计算新的生命值并触发<角色生命值UI更新>
    b. 如果当前角色hp=0，切换到<玩家死亡状态>

恢复生命值

1. 当前生命值加上传入的恢复生命值，需要注意需要用Clamp去修正一下当前结果，防止结果越界
2. 触发<角色生命值UI更新>

恢复饱食度

1. 当前饱食度加上传入的恢复饱食度，需要注意需要用Clamp去修正一下当前结果，防止结果越界
2. 触发<角色饱食度UI更新>

触发事件列表

1. 触发更新生命值事件, 当生命值发生变动时需要触发更新事件
2. 触发更新饱食度事件, 当饱食度发生变动时需要触发更新事件

以下为玩家武器相关方法

注意：需要记录当前使用武器数据和武器模型便于后续逻辑判断

修改武器: 武器数值、动画、图标等

1. 首先判断是否切换到新武器，如果没有切换武器则不需要进入逻辑，提升代码运行效率
2. 如果当前角色手上有武器则将之前的武器模型放入对象池
3. 查看新武器数据是否为空
    a. 如果为空代表玩家应该切换为空手状态并重置动画 `animator.runtimeAnimatorController = playerConfig.normalAnimatorController;`
    b. 如果不为空则需要将武器模型挂载在玩家右手处
        i. 获得武器配置数据
        ii. 根据配置实例化出武器的gameObject，将右手处设置为parent
        iii. 根据配置设置武器位置、角度、以及重置动画
    c. 将玩家状态切换为<玩家待机状态>

以下为战斗/伐木/采摘相关方法

检测地图对象能否受伤

1. 不同的地图对象对应武器类型不尽相同，需要检查武器类型是否合法
    a. 播放受伤音效
    b. 执行地图对象受伤方法

c. 更新武器成功攻击次数+1

武器触发：当武器碰到物体时（地图对象/AI对象）

1. 根据传入的Collider判断当前是否地图对象
    a. 记录当前攻击对象，在攻击动画结束前防止计算二次伤害
    b. 检测地图对象类型和武器类型是否合法
        i. <检测地图对象能否受伤>
2. 根据传入的Collider判断当前是否AI对象
    a. 获得武器配置信息
    b. 实例化<武器打击效果>对象并将其设置到碰撞体最近传入的参数点位置

```
// 显示打击效果
GameObject effect = PoolManager.Instance.GetGameObject(itemWeaponInfo.hitEffect);
effect.transform.position = other.ClosestPoint(currentWeaponGameObject.transform.position);
```

    c. 播放攻击音效
    d. 执行AI对象受伤方法
    e. 更新武器成功攻击次数+1

开始攻击

1. 清空攻击标记对象数值
2. 开启伤害检测`currentWeaponGameObject.transform.OnTriggerEnter(OnWeaponTriggerEnter);`

停止攻击

1. 清空攻击对象标记列表
2. 关闭伤害检测`currentWeaponGameObject.transform.RemoveTriggerEnter(OnWeaponTriggerEnter);`

攻击动作结束（攻击后摇）

1. 根据玩家攻击的地图对象数量更新武器耐久
2. 玩家状态切换到待机模式

受伤动作结束

1. 将玩家状态切换到默认状态（待机）

死亡动作结束

1. 调用GameManager中的方法结束掉整个游戏

玩家受伤

1. 判断当前玩家生命值是否>0，如果≤0则代表玩家死亡
2. 使用玩家当前生命值-=传入的伤害值
    a. 如果生命值≤0代表玩家死亡，切换到<玩家死亡状态>
    b. 如果生命值>0则需要<触发更新生命值事件>，并将玩家状态切换为<玩家受伤状态>

检查是否可以攻击当前地图对象

1. 检查：1. 角色当前是否可以进行攻击；2. 当前是否拿着武器；3. 当前武器类别是否与要求类别一致
2. 计算攻击方向，当进入攻击状态时会记录当前攻击方向，并且在Update时不断使玩家朝向这个方向，详情见<玩家攻击状态>
3. 播放武器配置中的音效
4. 切换状态至<玩家攻击状态>
5. 记录一下最后攻击对象，保证在攻击时只对单个对象生效

玩家选中地图对象/AI时的执行方法

1. 根据传入的射线检测结果进行判断，如果当前检测到地图对象
   `hitInfo.collider.TryGetComponent<MapObjectBase>(out MapObjectBase mapObject)`
    a. 判断玩家与当前地图对象的距离是否合法，如果距离过远则退出
    b. 判断当前地图对象是否可以拾取
        i. 如果当前地图对象可以拾取`mapObject.CanPickUp`并且鼠标已经按下，则查看当前地图对象配置的可拾取配置中对应的物品Id是否为合法，不合法则退出

ii. 如果物品能放回物品快捷栏

1. 拾取物品并销毁地图对象
2. 播放拾取动画和成功拾取音效

iii. 如果物品无法放回则弹出提示框提示背包已满

c. 判断当前地图对象是否可以进行攻击（伐木/采石）

i. 判断当前地图对象类型并检查当前武器是否满足地图对象类型要求，例如石斧只能用来砍树，详情见<检查是否可以攻击当前地图对象>

ii. 如果判断返回结果为false，则需要播放失败音效和弹出游戏提醒窗口

2. 根据传入的射线检测结果进行判断，如果当前检测到AI对象

`hitInfo.collider.TryGetComponent<AIBase>(out AIBase aiObject)`

a. 检查：1. 角色当前是否可以进行攻击；2. 当前是否拿着武器

b. 判断玩家与当前AI对象的距离是否合法，如果距离过远则退出，其中距离设定为武器攻击距离+AI半径

i. 计算攻击方向，当进入攻击状态时会记录当前攻击方向，并且在Update时不断使玩家朝向这个方向，详情见<玩家攻击状态>

ii. 播放武器配置中的音效

iii. 切换状态至<玩家攻击状态>

iv. 记录一下最后攻击对象，保证在攻击时只对单个对象生效

以下为辅助方法/生命周期函数

玩家状态切换

1. 根据传入的玩家状态类型切换玩家状态

a. 更新玩家标记变量，例如 canAttack/canUseItem/canPickUpItem

b. 使用状态机切换现在状态

播放音走路效

1. 调用音效管理器播放走路音效

播放动画

1. 根据动画名称播放动画

保存玩家数据

1. 保存当前玩家最后位置信息和转向角度
2. 保存当前玩家主要数据（生命值/饱食度）

Update

1. 检查游戏场景是否初始化成功
2. 实时计算玩家饥饿值和生命值<计算角色并更新当前饱食度>

OnDestroy

1. 销毁状态机

常见问题

玩家鼠标左键点击树木时发生的事情：

1. 输入系统：
    a. 玩家鼠标左键点击树木时Update会使用射线检查当前是否点击到了地图对象，如果点击到了则转交给玩家控制器处理，详情见（输入系统InputManager.cs）<检查鼠标选中地图对象是否可以进行互动>
2. 角色系统：
    a. 进入<玩家选中地图对象/AI时的执行方法>，检查前置条件是否符合（距离检查、能否攻击、武器类型是否正确等）
    b. <检查是否可以攻击当前地图对象>，如果可以攻击则计算玩家转向角度、播放音效、进入攻击状态
    c. 进入<玩家攻击状态>后会播放攻击动画，攻击动画中插入了事件帧Event，当动画帧数运行到标签时，Unity会触发Function对应函数（StartHit），执行Player_Model中StartHit对应的`OnStartHit`方法开启伤害检测

d. `OnStartHit` 会触发武器碰撞逻辑，武器碰撞逻辑中会检查当前武器和攻击对象之间的关系，当检查成功后会执行地图对象受伤方法并播放音效，此时树木会播放受伤动画并检查生命值，当生命值=0时掉落物品并跳转到地图管理器移除地图对象和生成掉落物品

3. 地图系统：

   a. 调用地图管理器移除方法移除树木，调用地图对象生成方法在原树木位置上生成掉落物品

4. 角色系统：与地图系统同步进行

   a. 当进入StopHit事件帧<停止攻击>：1. 关闭武器伤害检测；2. 并清空标记数组

   b. 当进入攻击动作结束事件帧<攻击动作结束>：1. 根据玩家攻击对象数量更新武器耐久度；2. 将玩家状态切换到待机

角色数据 Data/PlayerData.cs

```
[Serializable]
public class PlayerTransformData
{
    // 坐标: sv_postion存档用, position外部调用用
    private Serialization_Vector3 sv_position;
    public Vector3 position {
        get => sv_position.ConverToVector3();
        set => sv_position = value.ConverToSVector3();
    }

    // 旋转
    private Serialization_Vector3 sv_rotation;
    public Vector3 rotation {
        get => sv_rotation.ConverToVector3();
        set => sv_rotation = value.ConverToSVector3();
    }
}

// 玩家主要数据: 生命值、饱食度
[Serializable]
```

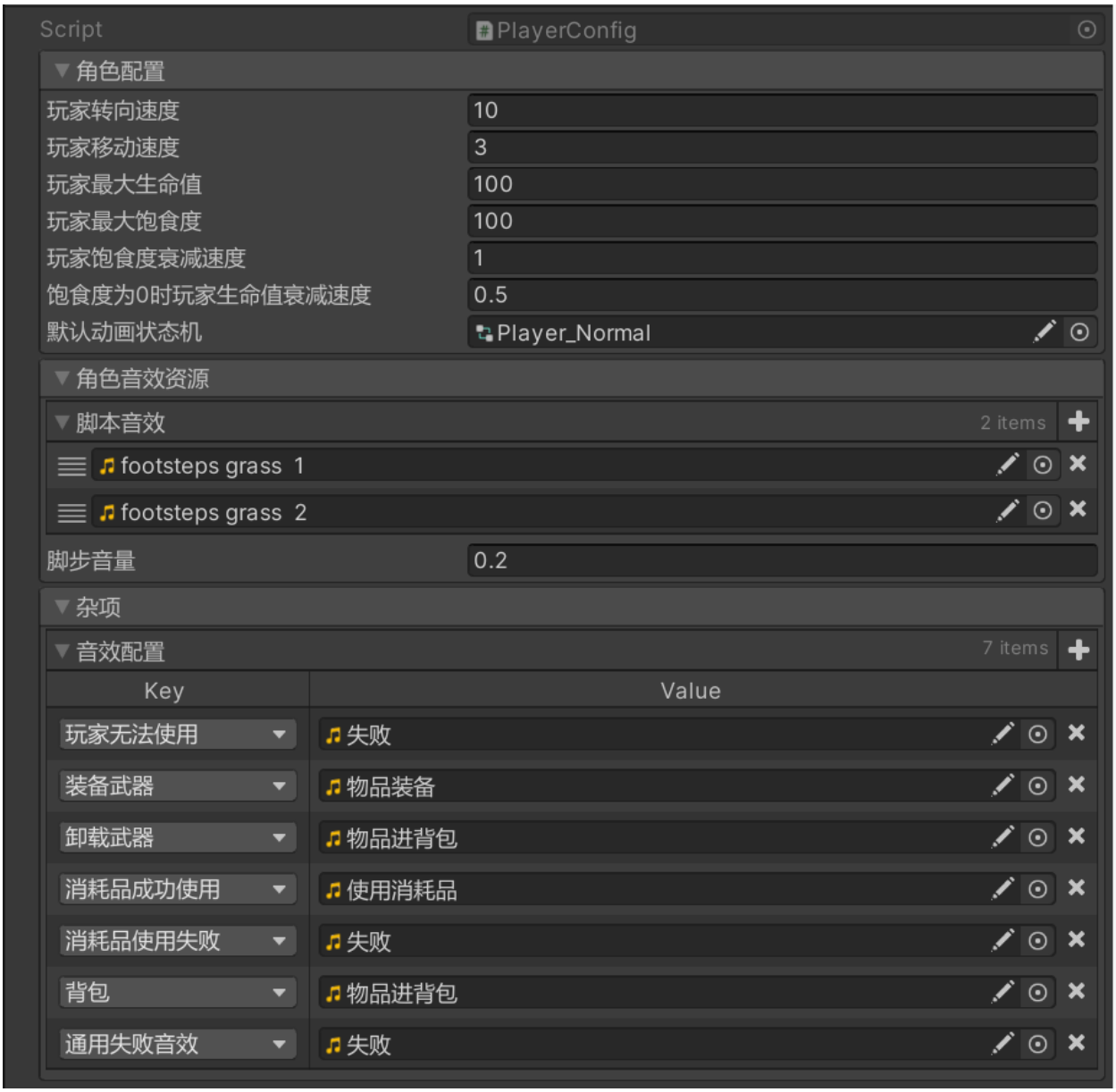
```
public class PlayerMainData
{
    public float hp;
    public float hungry;
}
```

角色配置 Config/PlayerConfig.cs

```
public class PlayerConfig : ConfigBase
{
    #region 角色属性配置
    [FoldoutGroup("角色配置"), LabelText("玩家转向速度")]
    public float rotateSpeed = 10;              // 玩家转向速度
    [FoldoutGroup("角色配置"), LabelText("玩家移动速度")]
    public float moveSpeed = 4;                 // 玩家移动速度
    [FoldoutGroup("角色配置"), LabelText("玩家最大生命值")]
    public float maxHP = 100;
    [FoldoutGroup("角色配置"), LabelText("玩家最大饱食度")]
    public float maxHungry = 100;
    [FoldoutGroup("角色配置"), LabelText("玩家饱食度衰减速度")]
    public float hungryReduceSpeed = 0.2f;        // 初设为5s掉1滴血
    [FoldoutGroup("角色配置"), LabelText("饱食度为0时玩家生命值衰减速度")]
    public float hpReduceSpeedOnHungryIsZero = 2.0f;
    [FoldoutGroup("角色配置"), LabelText("默认动画状态机")]
    public RuntimeAnimatorController normalAnimatorController;
    #endregion

    #region 角色音效资源
    [FoldoutGroup("角色音效资源"), LabelText("脚本音效")]
    public AudioClip[] footstepAudioClips;                // 脚步音效
    [FoldoutGroup("角色音效资源"), LabelText("脚步音量")]
    public float footstepVolume = 0.5f;                   // 脚步音效音量
    #endregion

    #region 杂项
    [FoldoutGroup("杂项"), LabelText("音效配置")]
    public Dictionary<AudioType, AudioClip> AudioClipDict; // 音效配置字典
    #endregion
}
```


Script
PlayerConfig
角色配置
玩家转向速度
10
玩家移动速度
3
玩家最大生命值
100
玩家最大饱食度
100
玩家饱食度衰减速度
1
饱食度为0时玩家生命值衰减速度
0.5
默认动画状态机
Player_Normal
角色音效资源
脚本音效
2 items
footsteps grass 1
footsteps grass 2
脚步音量
0.2
杂项
音效配置
7 items
Key
Value
玩家无法使用
失败
装备武器
物品装备
卸载武器
物品进背包
消耗品成功使用
使用消耗品
消耗品使用失败
失败
背包
物品进背包
通用失败音效
失败