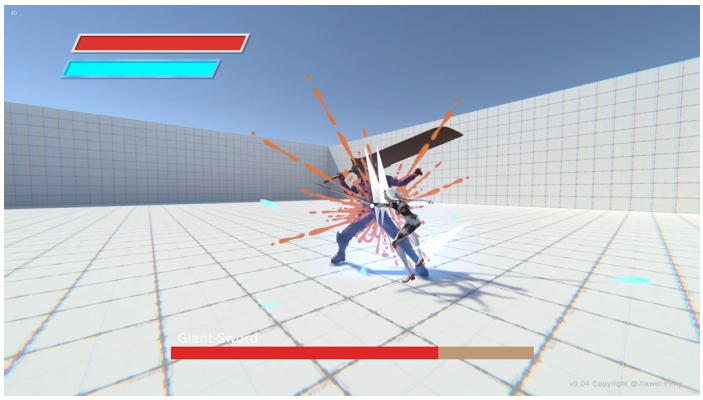
设计思考

关于弹反招式的表现/打击感设计

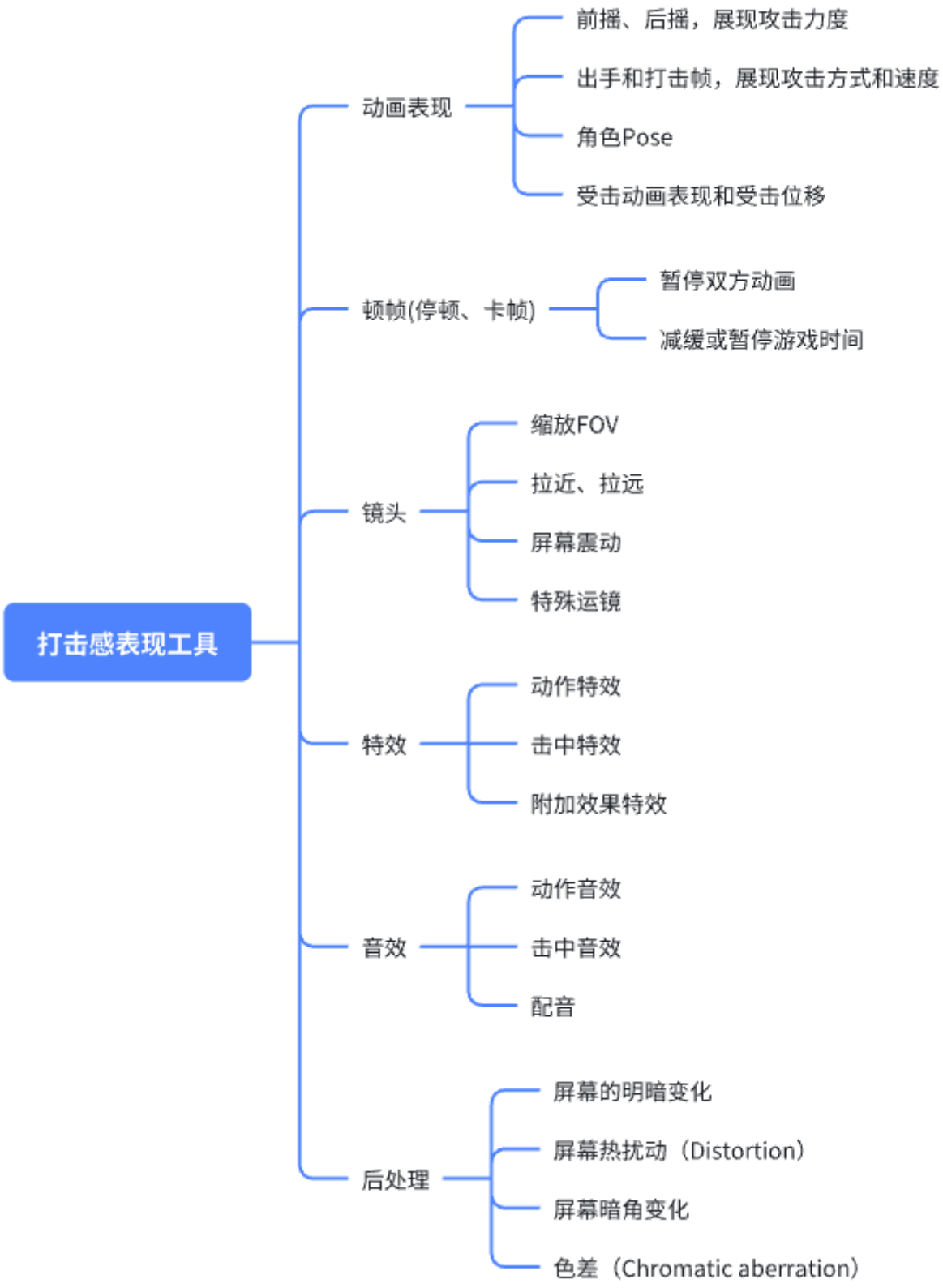


我希望针对描述一下游戏中，玩家进行**连续完美格挡→触发格挡弹反→派生弹反后追击二连**的这样一套连招的招式表现。我个人认为，战场的表现以及打击感等，一方面是为了服务于战斗爽感，但最核心的点是在于向玩家传达战斗过程中的信息。

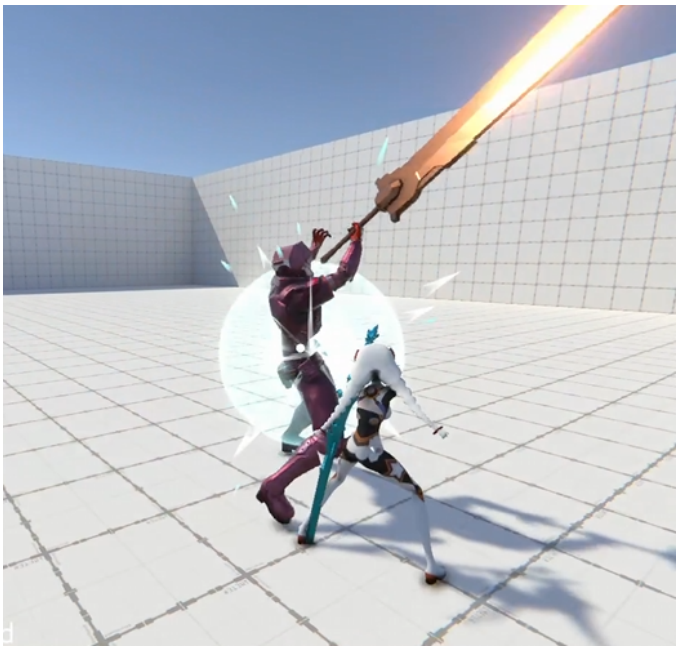
首先，在这一套连招当中，有以下几点信息需要通过表现来向玩家进行准确的传达：

1. 玩家/敌人 的攻击是否命中，以及 玩家/敌人 施加或受到的攻击的“力度”如何？无论这个攻击是被打到了 玩家/敌人 身上，还是被 玩家/敌人 成功格挡了。“力度”是该攻击的伤害值和削韧值的综合体现。（削韧值设计请参考系统策划案中的“*五、战斗机制详解→削韧机制*”）。因为这两个数值是目前我们这套战斗系统的最核心的，最关乎战场态势的数值。
2. 玩家是否成功弹反了敌人？弹反的一瞬间其实以为着玩家从防守节奏需要转变到进攻节奏，是一次非常关键的战场态势的转变，会显著的影响玩家的行为决策，需要明确的告知玩家。
3. 玩家是否完美格挡了敌人的攻击？没有完美格挡的玩家会陷入破防硬直当中，这会显著的改变玩家接下来的操作决策，并且会让玩家陷入被动扣血的不利状态，因此需要非常明确的告知。

对于打击感，我们主要可以用到以下的”工具“进行打击感表现：



接下来，我将详细讲述我是如何设计各个信息的打击感表现的：

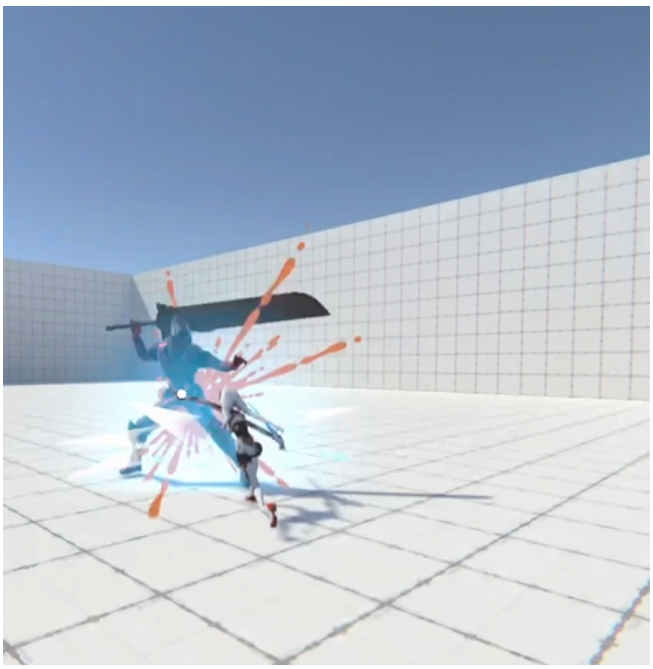


完美格挡

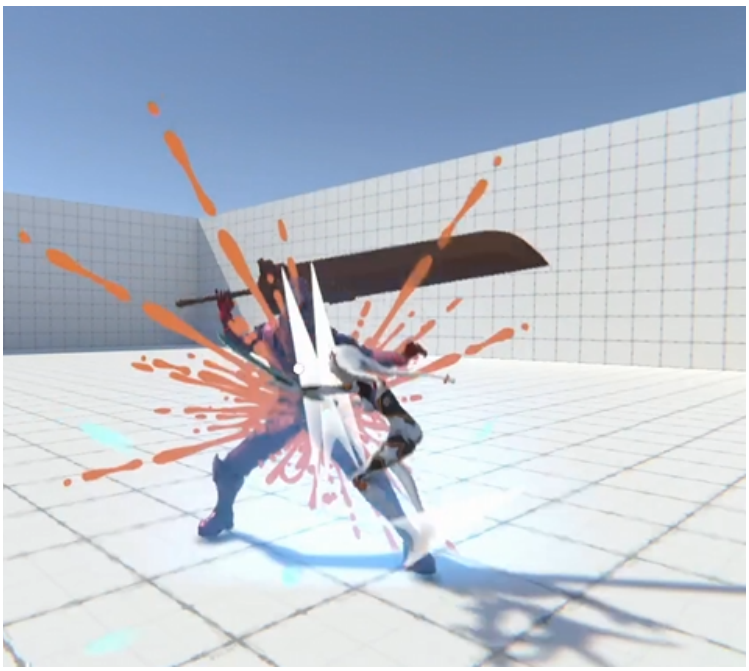
成功弹反

- **动画：**
  - 玩家角色通过摆出两种不同的Pose来展现二者不同的状态。
  - 被格挡的敌人会继续进行攻击的动作，而被弹反的敌人会进行特定的后撤硬直动作。前者动作连续进行，后者动作被强行打断并切换。前者动作倾向于靠近玩家，后者动作明显倾向于远离玩家。
- **顿帧表现：**
  - 完美格挡的顿帧时长与敌人攻击招式的“冲击力值”正相关，而“冲击力值”是依据敌人攻击的伤害值，削韧值，以及动作类型综合衡量进行配置的。（冲击力值设计请参考系统策划案中的“*五、战斗机制详解→冲击力机制*”）。但总体而言完美格挡的顿帧时间不会超过0.2秒。
  - 成功弹反会触发时停效果，其类似于顿帧，但时停效果长达1秒有余，并且为了避免时停时间过长打断整体战斗的连贯性，玩家在时停时可以随时进行攻击输入，就可以立刻打断时停进行攻击连招，流畅玩起来后，玩家可以通过快速输入完全跳过时停。时停设计的目的，首要的是向玩家传达攻防双方发生了互换这一信息。玩家由攻击方转为防守方，或由防守方转为攻击方的时刻，总会伴随着时停的出现。在这里，玩家由进行格挡防御的防守方，转为了要进行派生攻击的进攻方。
  - 玩家/敌人 没有被格挡的攻击，会依照其攻击的“冲击力值”决定顿帧的时长。其中，玩家在成功弹反后派生的二连击分别具有等级最高的“Heavy”和“Ultra Heavy”冲击力，同时伴随着明显的高额伤害。
- **镜头表现：**
  - 格挡与弹反
    - 在完美格挡时，镜头会根据玩家格挡的攻击的“冲击力值”决定震动的幅度、频率和时长。普遍在0.2秒以内。
    - 在成功弹反时，镜头会有一个持续时间更长的，振幅更大的镜头震动。

- 攻击中的镜头表现
  - 如下图所示，角色在弹反成功后派生的第一下追击攻击时，镜头会延迟的追上玩家。利用相对固定的镜头凸显玩家向前冲刺攻击的力度。而在第二段攻击时，镜头已经追上了玩家，此时第二段攻击的“冲击力值”和伤害都更高，镜头拉近后着重表现玩家第二段的Pose和击中后的顿帧、特效等效果。



弹反后派生攻击第一段



弹反后派生攻击第二段

- **特效表现：**
  - 格挡与弹反特效区分：
    - 完美格挡时触发的特效为较小的环形光圈，而成功弹反则以明显更大的光圈展现。这种大小上的对比能直观地让玩家感知到状态的转变。
  - 命中及伤害反馈：
    - 击中特效将精准标记出攻击落点，让玩家能立即确认攻击的准确位置。敌人被击中时，根据伤害数值高低，会触发不同程度的飙血效果，视觉上强化伤害反馈。
  - 色系区分：
    - 玩家相关： 由玩家触发的各项特效统一采用蓝色调，寓意其主动和正向收益。
    - 敌人相关： 敌人的行动特效则采用橙色调，方便玩家快速区分双方信息。
    - 受伤警示： 当玩家受伤时，飙血特效转为红色，以此作为明确的警示信号，强化负反馈效果。
- **音效表现：**
  - 音效调性与时长：
    - 完美格挡的音效相对短促，混响效果较弱，Pitch较低，表达出防御成功但攻击威力有限的信息。

        - 成功弹反的音效则设计为音频Pitch更高、持续时间更长，同时伴有明显的混响，传达出攻防转换的关键时刻和高收益感。
    - 负面反馈音效：
        - 格挡失败时，音效更为短促且低频，同时加入低频爆破音（尚未实现），以快速提醒玩家发生了错误操作，增强战场即时反馈感。
    - 进一步拓展：
        - 此外，针对攻击命中“肉”的质感，原计划设计了特殊的音效表现（尚未实现），旨在进一步丰富打击感和战斗真实度。
    - 设计理念：
        - 整体上，音频Pitch的高低直接对应玩家行为的成败：Pitch越高，表示行为越正确、收益越高；Pitch越低，则表明行为存在较大风险或负反馈。
- **后处理表现：**
    - 镜头Lens Distortion：
        - 对于正向收益行为（如攻击命中、完美格挡或成功弹反），镜头会产生由中心向外扩散的Lens Distortion效果，配合攻击的“冲击力值”来调整畸变幅度。
        - 反之，当玩家陷入负收益状态（受到攻击或格挡失败）时，则采用由中心向内收缩的Lens Distortion，视觉上营造出被动和受压的感觉。
    - Vignette（暗角）效果：
        - 正向行为时，画面外围会出现淡蓝色的暗角，与玩家蓝色系特效相呼应，传递玩家操作正确的信息。
        - 而负向行为则出现红色暗角，以警示玩家小心战斗。
        - 时停时，则会出现灰黑色的暗角，体现画面亮度被降低的感觉。（具体原因请看下文“*画面亮度调控*”）
    - 色差（Chromatic Aberration）：
        - 色差效果的强弱直接反映了行为收益的绝对值。无论是造成高额伤害还是遭受巨大打击，当数值较大时，画面会出现更显著的色差，强化视觉冲击力。
    - 画面亮度调控：
        - 在进入时停状态时，整体画面亮度会明显降低，突显当前节奏的“冻结”感。而每当玩家成功攻击命中时，画面会短暂地轻微提亮，用以快速反馈命中信息。

镜头处理设计

镜头锁定功能

核心作用： 保证玩家始终能聚焦于当前最重要的目标，无论是敌人、关键技能释放，还是防守反击的瞬间。

细节说明：

- 实现自动锁定敌人，使得镜头能随角色动态平滑切换，避免因过于频繁的镜头移动而丢失关键信息。
- 当战场中多个目标同时存在时，可设置优先级规则（如距离、与视角方向的夹角、攻击状态、伤害数值）来决定锁定目标。
- 结合锁定功能，可以在角色切换状态时提供短暂的缩放或震动效果，突出攻防转换的瞬间信息。
- 如果战场有多个敌人，锁定功能可以自动切换到有攻击意图的敌人身上，降低玩家对于操作镜头的负担。（原设计方案中包含玩家同时与多名敌人战斗的设计，其中就实现了自动切换锁定目标的功能）

针对不同动画和战场状态的分类管理

核心作用： 根据角色状态（如攻击、防守、被击、技能施放等）、攻击动画（如普通战斗、闪避、连击，突进攻击），以及战场状态（敌人数量，敌人距离，敌人体型大小和攻击方式，敌人相对位置等）调整镜头表现。

细节说明：

- 为不同的角色状态设计不同的镜头效果：

    - **攻击阶段：** 采用动态追踪与拉近效果，突出打击瞬间的力度。同时，也可以让镜头配合攻击的动作进行相应的移动。比如角色做出一个向前刺击的攻击动作，镜头也可以跟随着动作一起向前突刺，强调该动作的力量感，传到动作成功触发和命中的信息，可以用于处决动画。
    - **防守阶段：** 利用轻微拉近和局部特写，展示角色防御动作的细节，同时降低震动幅度，突出“稳固”感。
    - **技能释放：** 配合特殊的镜头特效（如慢动作、震动、镜头模糊）强调技能的爆发力和战场转折。
    - **特殊大招：**可以制作单独运镜的镜头轨道动画来表现特殊大招。
- 为不同的战场状态设计不同的镜头效果：
    - **当有多名敌人时：**可以拉远镜头或适度扩大FOV，让画面中包含更多的敌人。同时可以降低镜头外敌人的攻击欲望，让玩家可以专注于和眼前的敌人战斗。
    - **当玩家与敌人接近时：**调整镜头的视角，适度拉远镜头让敌人和玩家角色同时处于画面当中，并且尽可能不让玩家角色和敌人产生遮挡。（可参考下文的*镜头视角*）
- 为不同的环境设计不同的镜头效果：
    - 当玩家在狭窄的空间时，需要拉近镜头，采用贴近的越肩视角。
    - 当玩家来到开阔的区域时，可以拉远镜头，展现空间感。

为镜头制作轨道动画

核心作用： 预设一定的轨迹和运动规律，使镜头在特定战斗阶段中实现平滑过渡，突出关键动作。

细节说明：

- 根据战斗场景设计不同的轨道动画，例如在攻击连招中，镜头可以按照预设轨迹追踪玩家或敌人的动作；
- 在一些重要技能触发时，镜头轨迹可临时切换为特效镜头，快速拉近或旋转，强调技能效果和战场紧迫感。

针对目标设备进行镜头远近和FOV的微调

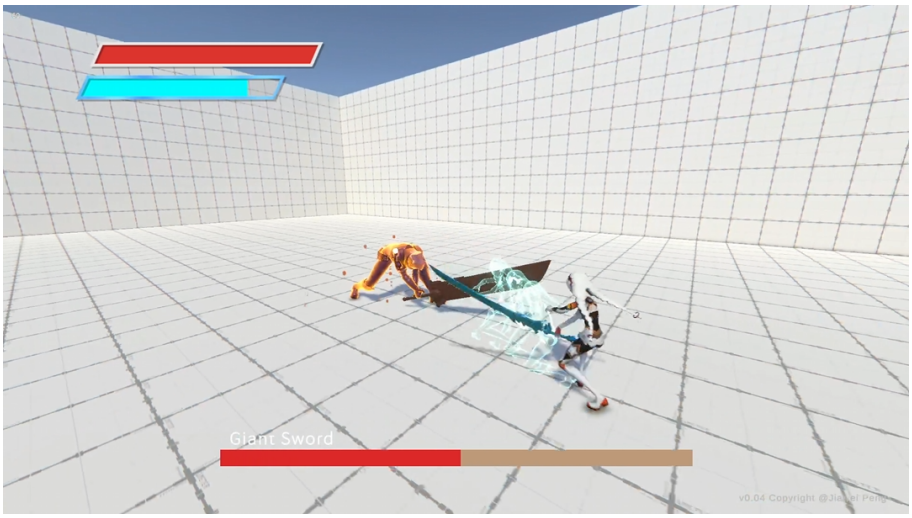
核心作用： 保证不同平台或显示设备下均能呈现最佳的视觉效果和战斗信息。

细节说明：

- 根据设备的屏幕尺寸、分辨率以及操作方式（PC、主机、移动端）进行镜头参数的预设调整，确保视野范围（FOV）和画面远近的最佳平衡；
- 移动端可能需要更靠近的镜头以增加视觉冲击，而大屏设备则可以利用更宽的视角展示更多战场信息；

- 动态调整FOV，例如在关键时刻临时缩放镜头，突出重要动作，同时保证信息不丢失。

镜头视角

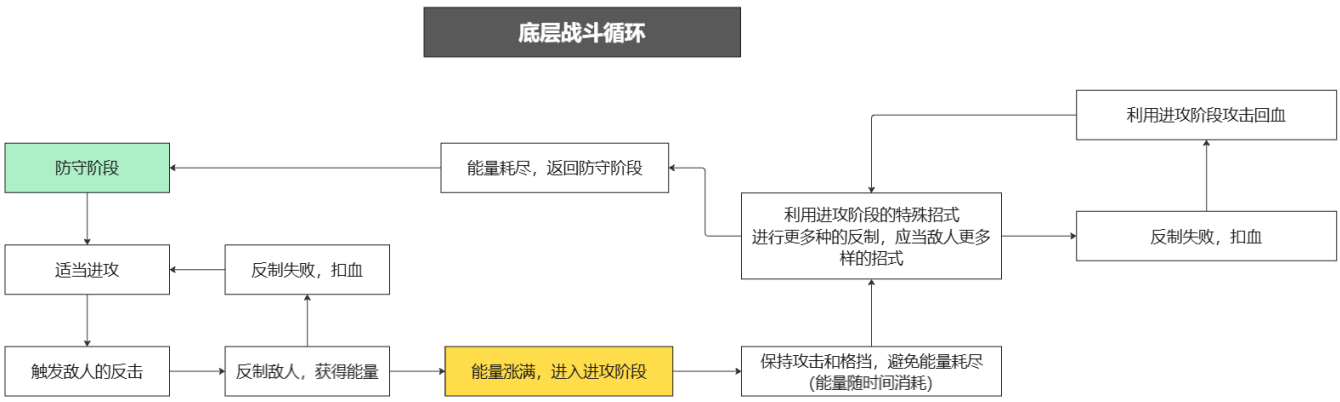


核心作用： 在本Demo中，游戏采用了略微偏向一侧的视角，目的是为玩家提供更宽阔的战场视角，避免正面视角下玩家角色遮挡敌人角色的动作或特效表现从而影响信息传递的问题。

细节说明：

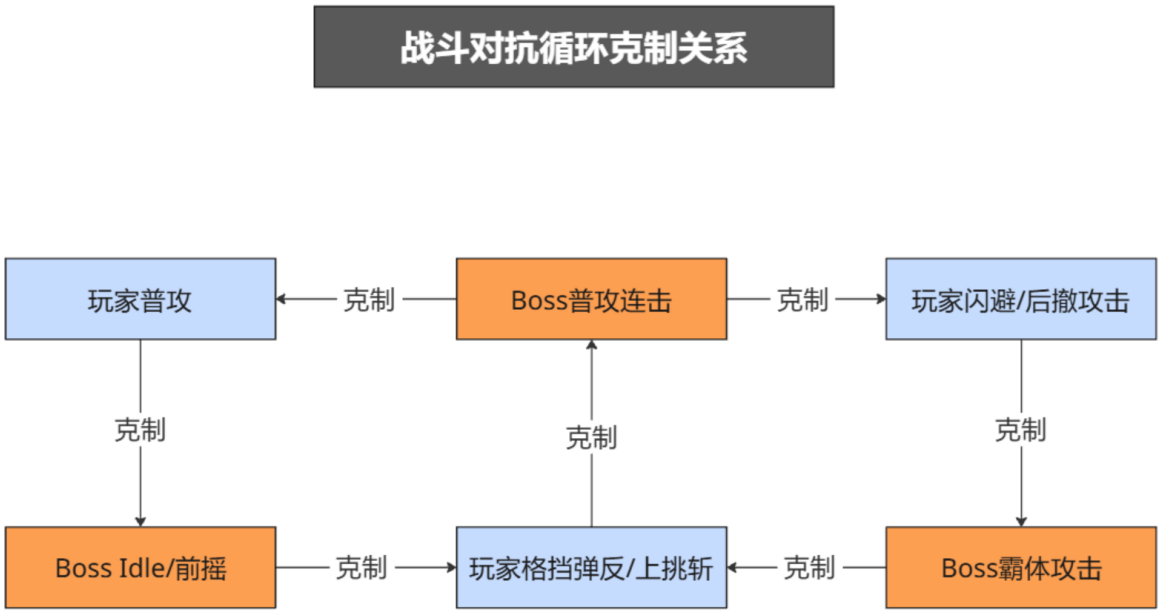
- 采用侧视视角，使玩家能够同时看到敌人和自己在空间中的相对位置，有助于玩家进行距离判断。
- 玩家可以更加清晰的看到敌人的动作，尤其是小幅度的前摇动作，也可以避免被玩家角色的身体遮挡从而阻碍玩家获取信息。

Demo的底层战斗机制



角色与怪物如何形成机制对抗

敌人与玩家的对抗机制主要是由各种招式的克制关系构成的。这种循环的克制关系可以让玩家有基本的行为决策判断的方向，同时创造出与Boss的博弈感和交互感。以下是一个最简化的战斗对抗的克制关系图，去除了玩家角色和Boss具体的招式设计：



谁更具有战斗优势

如果单纯从上图可以看出似乎Boss更具有战斗优势，因为玩家的“玩家格挡弹反/上挑斩”这一项被两个Boss的行为克制。但实际上，玩家在游戏中可以利用普攻逼迫Idle或前摇状态的Boss开始格挡，格挡一定次数后的Boss会强制进入普攻连击状态，此时玩家就可以利用格挡弹反来应对Boss的普攻连击，并且在成功应对之后，可以触发Boss的大硬直。因此单纯从机制的角度来讲的话，玩家其实具有多种应对Boss攻击的手段，并且可以通过角色的出招和行为来强迫或勾引Boss做出对应的其他对玩家有利的行为，从而将Boss的行为引导到对玩家自身有利的状态下来。并且因为该Boss是游戏中前期定位的Boss，希望玩家可以利用它逐步掌握引导AI出招的能力，因此这个Boss AI被设计的较为简单就可以被引导出招，只需要玩家进行最符合直觉的普通攻击，就可以将Boss引导到对自己有利的状态下来，从而可以开始使用格挡弹反等技能来创造Boss硬直。

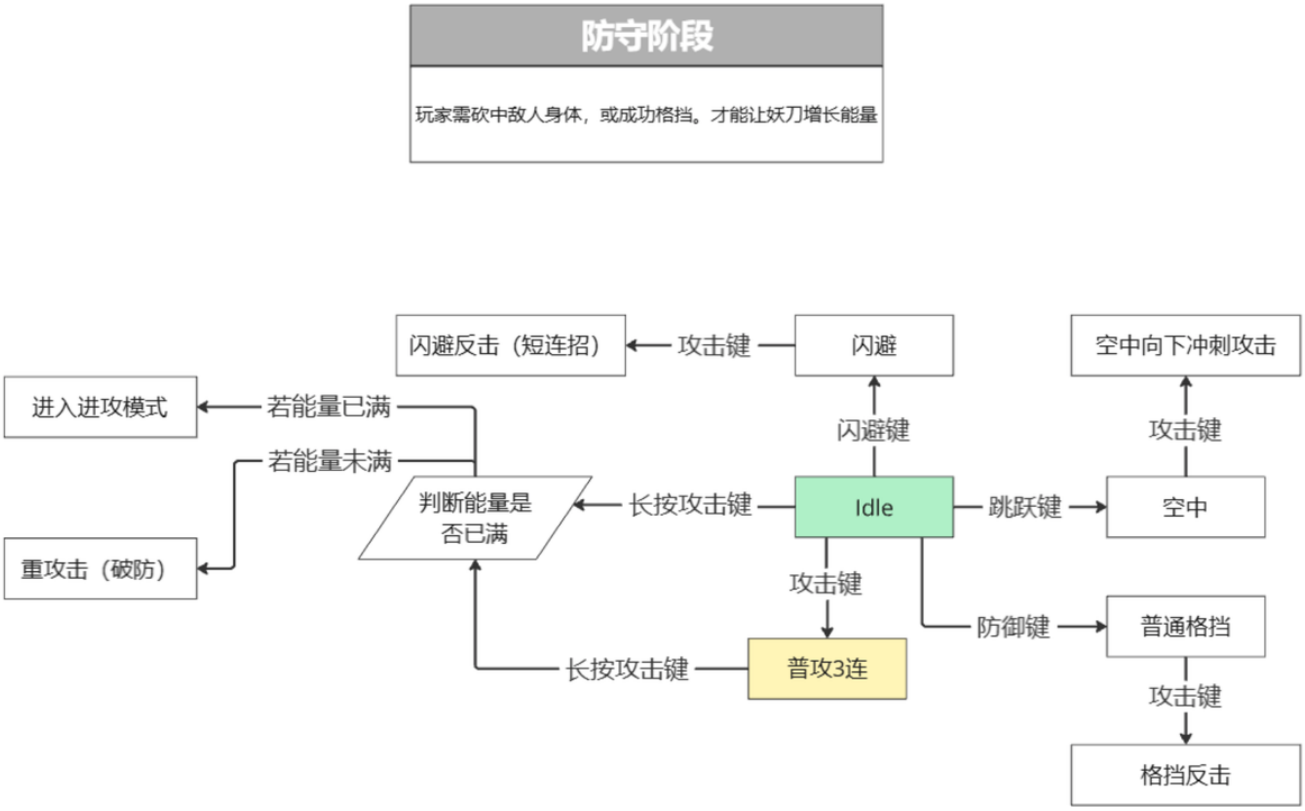
战斗的核心乐趣点

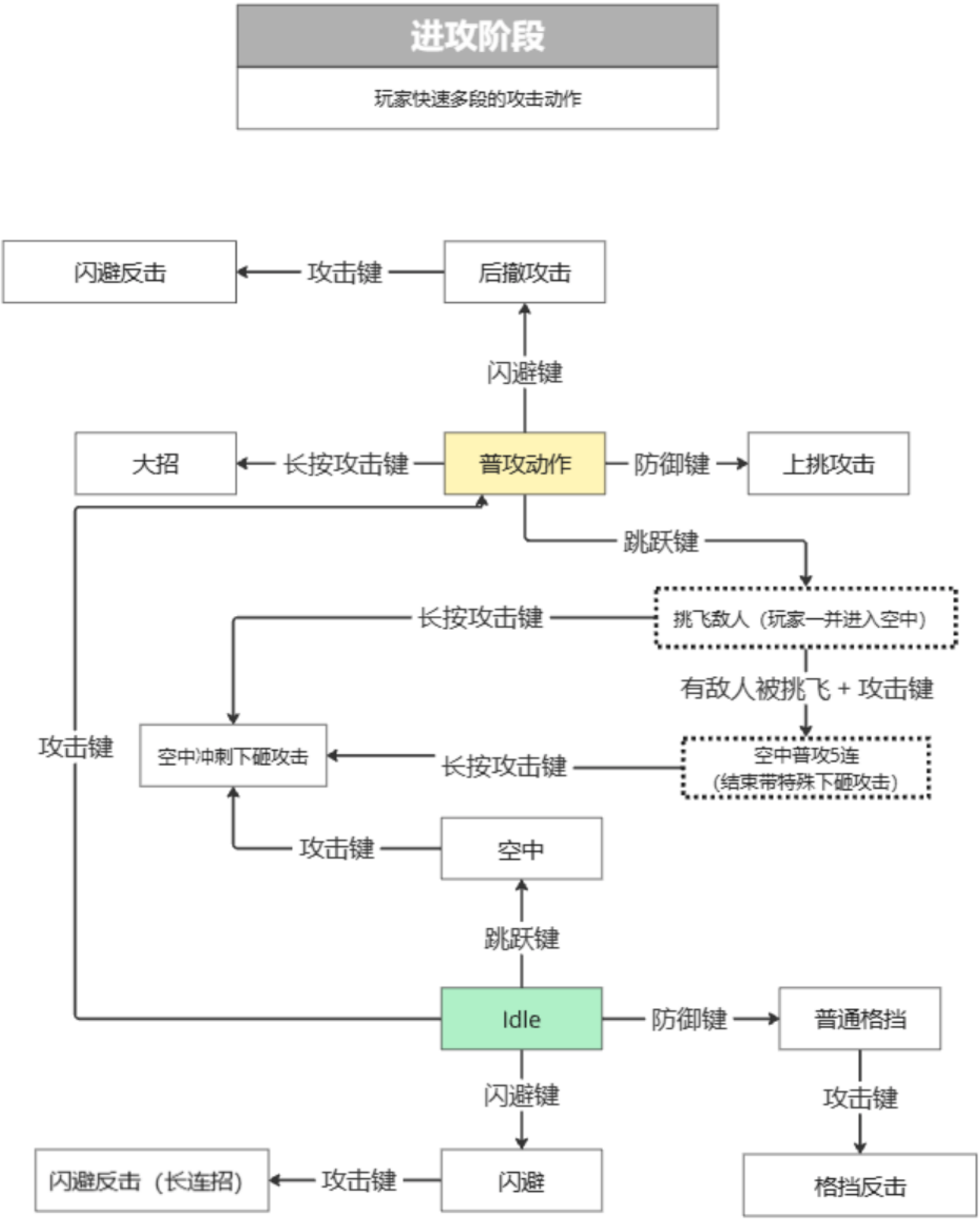
这个战斗的核心乐趣点是期望玩家通过与Boss对战逐步了解到这种Boss和玩家之间招式的对抗克制关系。以及BossAI在不同情况下会有何种表现，做出何种招式，随后玩家可以在进攻状态下一边攻击，一边衔接可以化解或克制Boss各种不同攻击行为的特殊招式，与Boss进行不间断的立回的乐趣。

动作衔接关系，以及角色操作手感优化

动作衔接关系

这是当前游戏中，玩家的各个招式动作之间的衔接关系图：



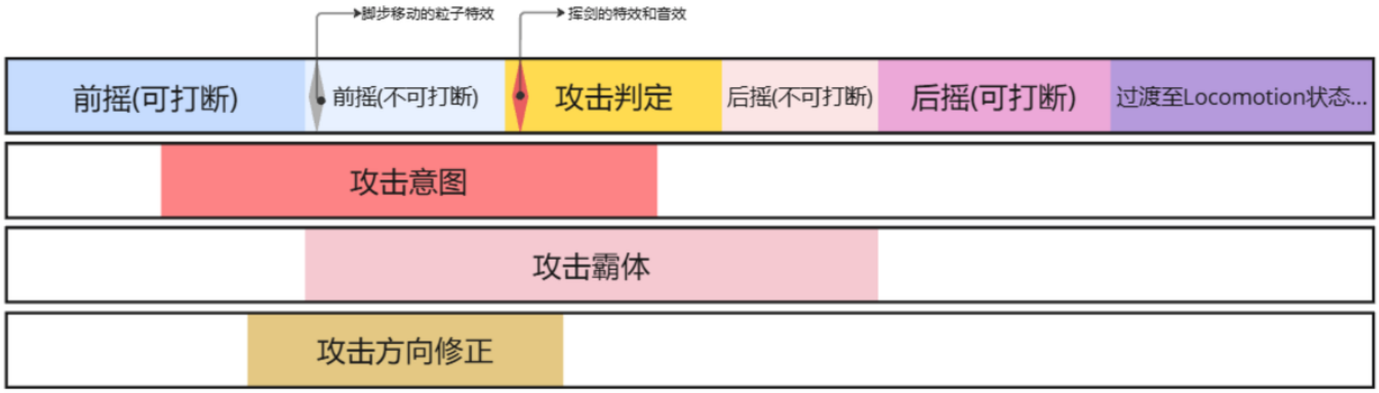


虚线框框为未实现的内容

优化操作手感

攻击时间划分：

为了优化角色的操作手感，游戏内的玩家角色和敌人角色都采用了以下的逻辑，来划分攻击的可用时间：



前摇（可打断）：在此状态，玩家可以通过**闪避和格挡**来打断当前的攻击动作。
前摇（不可打断）：在此状态，攻击动作不能被玩家输入打断。
攻击判定：在此期间，系统开启对应的攻击Hitbox。
后摇（不可打断）：此时，Hitbox关闭，玩家输入不能打断攻击动作。
后摇（可打断）：此时，玩家可以通过**闪避**来打断当前的攻击动作，也可以通过按下连招对应的攻击按键衔接下一段攻击动作。
过渡至Locomotion：此时，动画将过渡至Locomotion，玩家可以进行任何输入行为。连招记录被清空，此时按下攻击按键将从头开始连招。
攻击意图：将在玩家前方开启攻击意图判定区域。敌方在自由状态下将会试图格挡和闪避玩家的攻击。敌人也会启动攻击意图判定区域，玩家在区域内闪避将会触发成功躲避攻击的完美闪避。
攻击霸体：在此期间，玩家具有防御性能提升，可以抵抗更强的冲击力而不中断动作，同时具有霸体减伤。
攻击方向修正：在此期间，敌人和玩家可以修正自己的朝向。

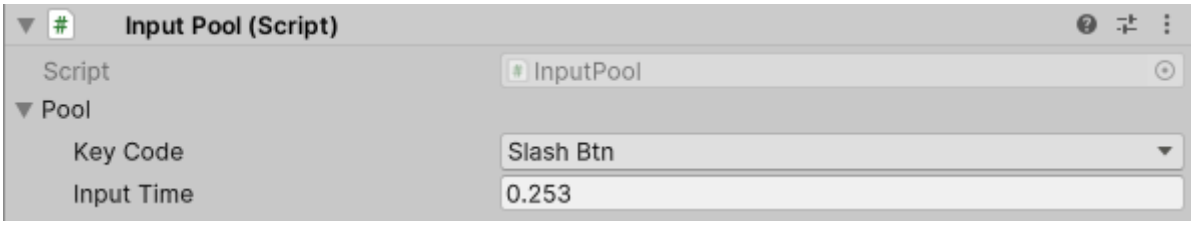
时间长度仅为示意，根据不同的攻击动画有不同的时长

通过将一次攻击细分为多个阶段，我们能更精细地控制角色动作的响应性和中断时机，从而提升操作手感。具体来说：

- **前摇阶段（可打断与不可打断）：** 为玩家提供了明确的输入窗口，既能通过闪避和格挡打断攻击，也确保关键动作不会轻易被打断，从而平衡自由度与连贯性。
- **攻击判定阶段：** 开启Hitbox确保攻击命中时有即时反馈，使玩家能清晰感知攻击效果，增强打击感。
- **后摇阶段（可打断与不可打断）：** 让玩家在技能释放尾声有机会衔接连招或闪避，既延续了攻击流畅性，又保持了操作的灵活性。
- **过渡至Locomotion：** 明确的动画过渡区间使角色动作自然结束，同时重置连招记录，避免了操作混乱。
- **攻击意图、霸体与方向修正：** 提供了预判和保护机制，既允许玩家调整攻击方向，也增加了战略性的操作反馈，从而使战斗更具层次感。

输入缓冲 Input Buffer

游戏提供了一个输入缓存系统，可以将玩家按下的输入指令存入一个缓存池中，缓存池只会保留最新的一次输入，并且只会保留0.2s。



比方说，当玩家在游戏还没有允许进行衔接攻击时就按下了攻击键，攻击键会被暂存0.2s。当0.2s内玩家操控的角色进入了可以进行衔接攻击的状态时，它就会识别到玩家已经进行了输入并立即开始衔接攻击。这样玩家的输入就不必非常的精准，也可以成功判定。

土狼时间 Coyotes Jump

虽然本Demo不包括考验跳跃的环节，但依然包含了用于提升手感的土狼时间相关的代码，其允许角色在离开地面或平台只会的一小段时间内，按下了跳跃按键后依然可以起跳。这避免了角色可能因为判定上的严格，使得玩家虽然视觉上觉得自己并没有离开平台，但却已经滞空导致无法起跳的情况。

```
//检查滞空时是否再次按下了空格按键
if (player.inputPoolMgr.pool.KeyCode == InputButtonType.Jump)
{
    player.inputPoolMgr.pool.Clear();
    if (coyotesJumpTime >= Time.timeSinceLevelLoad)
    {
        player.EnterJumping();
    }
}
```

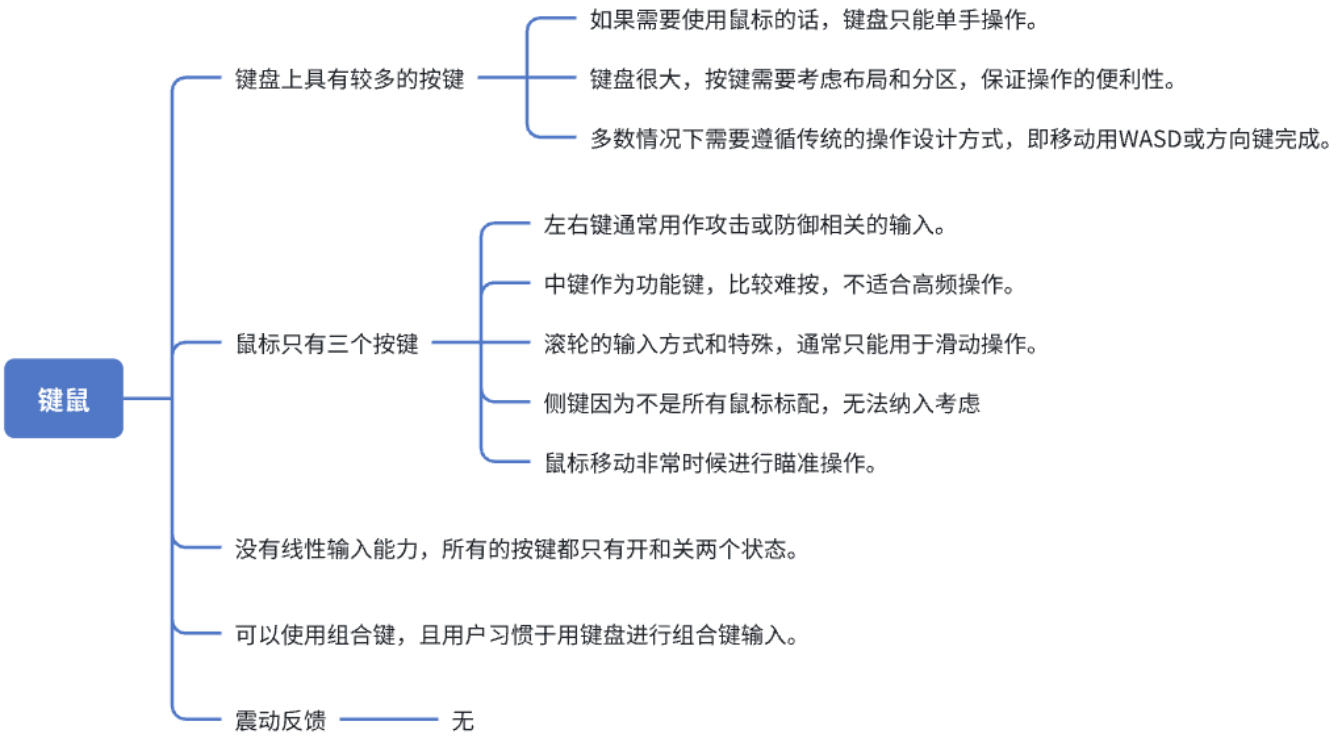
边缘保护 Edge Protect

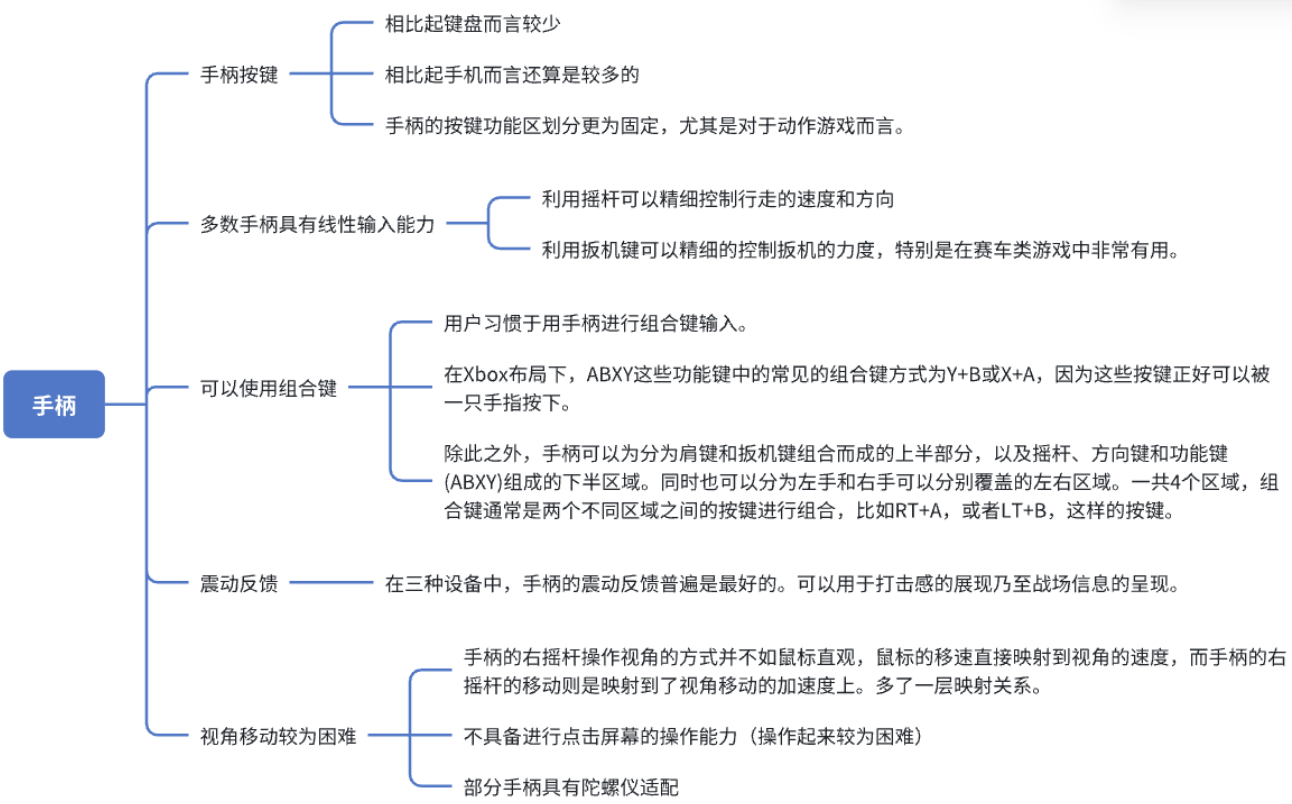
```
        //边缘保护
        public void EdgeProtect(float protectForceMultiplier = -1f, float pushBackForceMultiplier = -1f)
        {
            if (!isOnGround) return;

#if UNITY_EDITOR
            rayDatas.Clear();
#endif
            notFoundEdgeCount = 0;

            for (int i = 0; i < EdgeCheckNum; i++)
            {
                m_detectDirection = MathC.AngleToDir(i * m_EdgeCheckStep);
                m_startPoint = transform.position + m_detectDirection * EdgeRadius + transform.up;

                if (Physics.Raycast(m_startPoint, transform.up * -1, 1.5f, playerData.GroundLayer))
                {
#if UNITY_EDITOR
                    rayDatas.Add(new RayData(true, m_startPoint, m_startPoint + transform.up * -1 * 1.5f, Vector3.zero));
#endif
                }
                else
                {
                    edgeCheckResult[notFoundEdgeCount] = m_detectDirection;
                    notFoundEdgeCount++;

#if UNITY_EDITOR
                    rayDatas.Add(new RayData(false, m_startPoint, m_startPoint + transform.up * -1 * 1.5f, m_removedSpeed));
#endif
                }
            }

            //如果没有检测到任何地面，则视为在空中
            if (notFoundEdgeCount >= EdgeCheckNum) return;

            for (int i = 0; i < notFoundEdgeCount; i++)
            {
                m_detectDirection = edgeCheckResult[i];
                m_removedSpeed = m_detectDirection * Vector3.Dot(Rig.velocity, m_detectDirection)
                    * (protectForceMultiplier > 0f ? protectForceMultiplier : EdgeProtectForceMultiplier);

                // 将被移除的速度分量转移到其他方向
                Vector3 transferSpeed = m_removedSpeed - m_detectDirection * Vector3.Dot(m_removedSpeed, m_detectDirection);
                Rig.velocity += transferSpeed;

                if (Math.Sign(Rig.velocity.x) == Math.Sign(m_detectDirection.x))
                {
                    //消解玩家在x方向上的移动分量
                    Rig.velocity -= new Vector3(m_removedSpeed.x, 0, 0);
                }

                if (Math.Sign(Rig.velocity.z) == Math.Sign(m_detectDirection.z))
                {
                    //消解玩家在z方向上的移动分量
                    Rig.velocity -= new Vector3(0, 0, m_removedSpeed.z);
                }

                //缓慢的将玩家推离边缘
                Vector3 pushAwayDirection = -m_detectDirection.normalized;
                Rig.velocity += pushAwayDirection
                    * (pushBackForceMultiplier > 0f ? pushBackForceMultiplier : EdgePushBackForceMultiplier); // 调整0.1f以控制推离边缘的力的大小
            }

            if (Rig.velocity.y > 0 && notFoundEdgeCount > 0) Rig.velocity = Rig.velocity.ZeroY();
        }

        #endregion
```

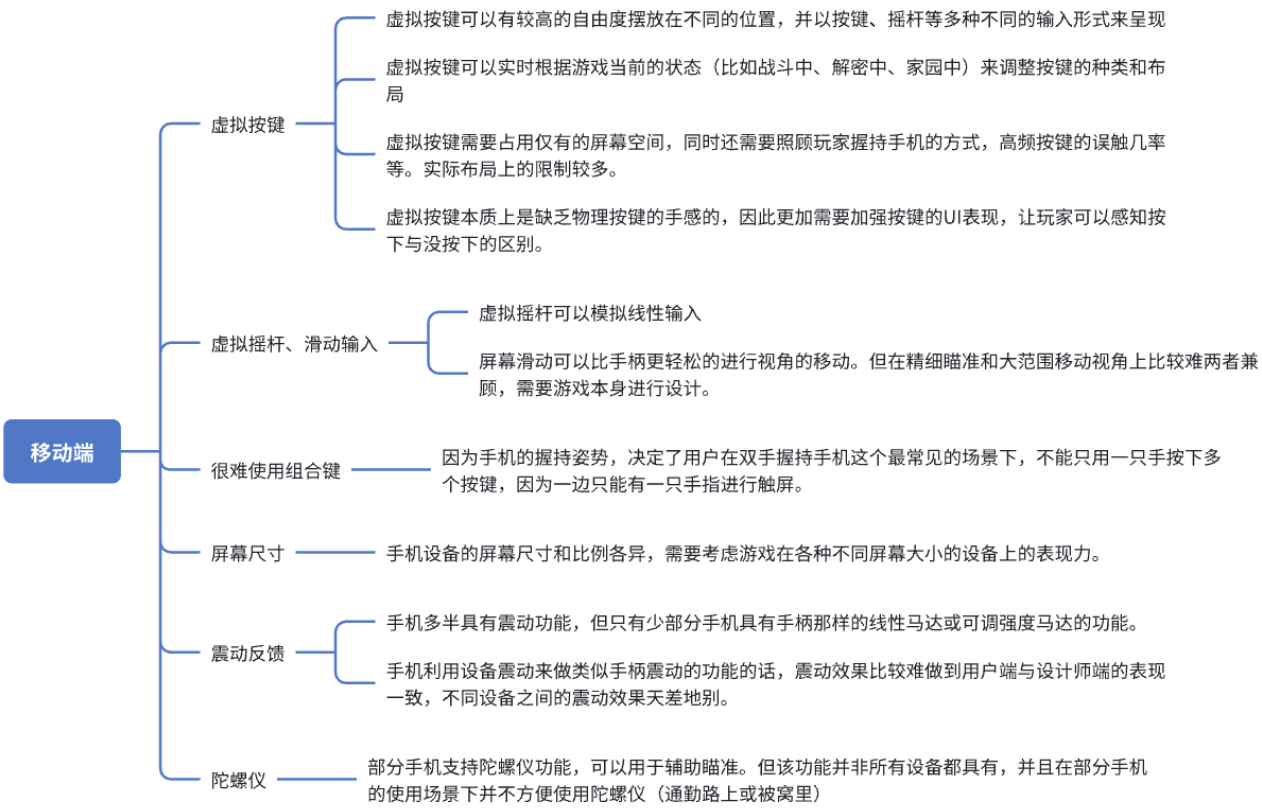
当角色因为攻击、闪避等动作产生了位移的时候，本Demo会开启边缘保护功能。可以自动的避免角色因为其他动作产生的位移而跌落平台。而当玩家移动时，依然可以正常的离开平台。

多种不同输入设备的兼容优化

在回答问题之前，我先总结了三种不同的输入设备上的共同点和互为优劣的地方：


键鼠
键盘上具有较多的按键
如果需要使用鼠标的话，键盘只能单手操作。
键盘很大，按键需要考虑布局和分区，保证操作的便利性。
多数情况下需要遵循传统的操作设计方式，即移动用WASD或方向键完成。
鼠标只有三个按键
左右键通常用作攻击或防御相关的输入。
中键作为功能键，比较难按，不适合高频操作。
滚轮的输入方式和特殊，通常只能用于滑动操作。
侧键因为不是所有鼠标标配，无法纳入考虑
鼠标移动非常时候进行瞄准操作。
没有线性输入能力，所有的按键都只有开和关两个状态。
可以使用组合键，且用户习惯于用键盘进行组合键输入。
震动反馈
无


手柄
手柄按键
相比起键盘而言较少
相比起手机而言还算是较多的
手柄的按键功能区划分更为固定，尤其是对于动作游戏而言。
多数手柄具有线性输入能力
利用摇杆可以精细控制行走的速度和方向
利用扳机键可以精细的控制扳机的力度，特别是在赛车类游戏中非常有用。
可以使用组合键
用户习惯于用手柄进行组合键输入。
在Xbox布局下，ABXY这些功能键中的常见的组合键方式为Y+B或X+A，因为这些按键正好可以被一只手指按下。
除此之外，手柄可以为分为肩键和扳机键组合而成的上半部分，以及摇杆、方向键和功能键(ABXY)组成的下半区域。同时也可以分为左手和右手可以分别覆盖的左右区域。一共4个区域，组合键通常是两个不同区域之间的按键进行组合，比如RT+A，或者LT+B，这样的按键。
震动反馈
在三种设备中，手柄的震动反馈普遍是最好的。可以用于打击感的展现乃至战场信息的呈现。
视角移动较为困难
手柄的右摇杆操作视角的方式并不如鼠标直观，鼠标的移速直接映射到视角的速度，而手柄的右摇杆的移动则是映射到了视角移动的加速度上。多了一层映射关系。
不具备进行点击屏幕的操作能力（操作起来较为困难）
部分手柄具有陀螺仪适配



因此，我认为在多端同步游戏时，要在多个设备上保证体验的相近，需要遵守以下原则：

- **一致性反馈：**
  - 不论是视觉、听觉还是震动反馈，必须确保不同设备传达的信息能让玩家获得相似的操作感受和成就反馈。例如，在技能释放时都能感受到明确的打击感、时停效果或镜头震动，帮助玩家判断技能是否成功、释放是否及时。同时，镜头远近、画面表现和特效、乃至人物动作的动画幅度等，都会受到玩家设备的屏幕大小的影响。可以考虑根据不同的平台，针对性的调整上述的内容，确保在最小的手机屏幕上也可以准确的传达角色技能的前摇、释放、命中、产生效果、后摇等。
- **统一操作逻辑：**
  - 尽量保证技能输入逻辑在各平台上保持一致，让玩家在跨平台切换时不必重新适应操作。可以采用灵活映射和自定义按键（或布局）来缓解不同设备硬件限制带来的差异。

需要规避的设计误区

1. **过度依赖组合键输入方式：**
   避免在技能释放上过度依赖键盘复杂组合或手柄特定按键组合，确保在触屏上也能方便操作。
2. **忽视反馈一致性：**
   不应只在某一平台的特性（如手柄的震动）上做文章，而忽略了其他设备（如键鼠或触屏）的反馈体验，导致玩家在不同设备上体验差异巨大。而更需要着眼于各个平台共同的视觉表现。

3. **不合理的按键布局：**
   针对触屏，切勿设计过于密集的虚拟按键区域；针对键鼠，不合理的组合键设计会影响单手操作鼠标时的连贯性。同时，像黑魂和法环就出现了一个经典的没有考虑键盘适配，而只考虑了手柄适配的按键输入布局方式。其游戏具有四种不同的攻击动作，正好对应了手柄上部分的肩键+扳机键，而在键盘上，则是使用了Shift+鼠标左键右键的形式，将一个特别高频的操作作为了组合键。
4. **视角与操作不匹配：**
   例如手柄右摇杆映射视角控制不应与键鼠直接等效，设计时需要针对每种设备调整视角移动的灵敏度与加速度，避免影响技能释放时的瞄准和视觉反馈。避免设计需要极度精确瞄准的技能，如果必须设计，可以为手柄和手机提供一定程度的辅助瞄准功能。