实现射击游戏操作面板

在“创建他人可见的人物模型”的基础上实现下列功能。

换弹动作实现

如图 1，新建一个输入操作 IA_Reload，将其触发器设置为已按下。并在 IMC_Weapons 内添加一个新的映射，选择 IA_Reload 且按键映射为 R。

图 1 添加换弹映射

在 Weapons 目录下的蓝图接口 CombatInterface 内创建 Reload 函数。如图 2，在 BP_FirstPersonCharacter 蓝图内调用刚刚添加的 Reload 函数发送消息，实现换弹事件。

图 2 换弹事件

如图 3，在 BP_Gun 的事件内实现接口内的 Reload 方法调用 SeverReload 事件。

图 3 实现 Reload 事件的基本调用

其中 SeverReload 事件的实现如图 4 所示，这里需要将 SeverReload 的复制属性修改为仅在服务器运行。

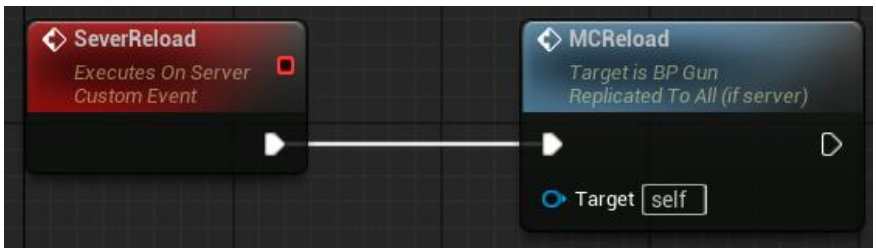

图 4 实现 SeverReload 事件的基本调用

其中 MCReload 事件的实现如图 5 所示，其复制属性设置为组播。这里为第三人称设置的动画蒙太奇是由动画资产包 AnimStarterPack 内的 Reload_Rifle_Hip 生成，第一人称射击动画需要自行创建。

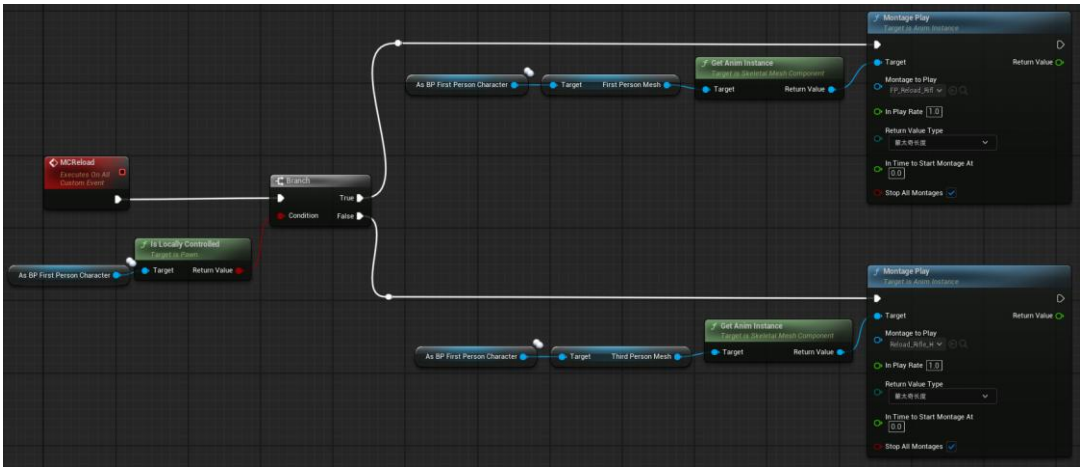

图 5 实现 MCReload 事件

这里给出第一人称射击动画的制作方法：将 Reload_Rifle_Hip 复制一份保存为 FP_Reload_Rifle_Hip，然后进入动画将其导出为.fbx 文件，在、z 轴旋转-90°后重新导入，进入其骨骼界面，切换到重定向面板将第一人称的骨架添加到兼容骨架内，之后返回动画界面，将其网格体替换为第一人称骨骼网格体，使用创建好的动画资源创建动画蒙太奇（如图 6）。

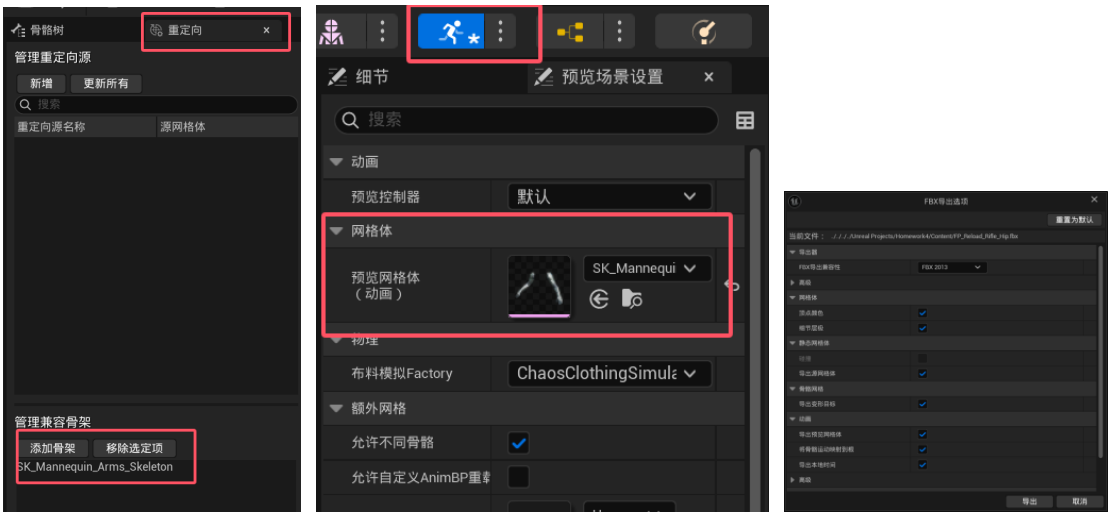

图 6 兼容第一人称骨架并将其应用到第一人称骨骼网格体

如图 7 在动画插槽管理器中添加 Arms 插槽，然后将图 6 创建的动画蒙太奇的插槽替换为 DefaultGroup Arms。

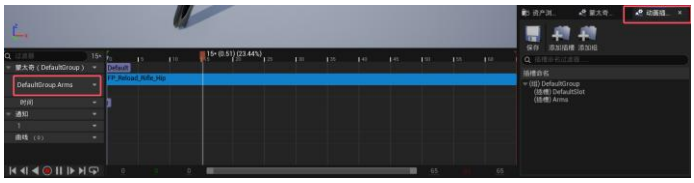

图 7 修改动画插槽与射击动画相同

子弹数量 UI

在内容->UI->GameUI 新建一个蓝图类继承 HUD 类，命名为 InGameHUD 用于管理 HUD 的 UI。进入 BP_FirstPersonGameMode 将 HUD 更改为刚刚创建的 InGameHUD。同样在 GameUI 内创建一个控件蓝图 PlayerInGameHUD 用于后续展示 UI。

如图 8，制作一个用于显示子弹数量的控件蓝图。

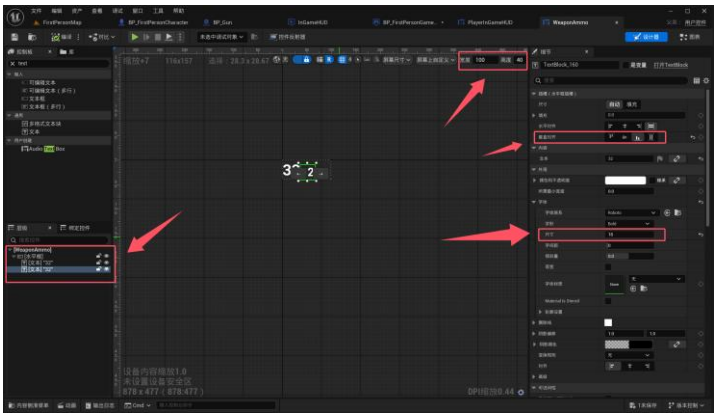

图 8 子弹数量控件蓝图

如图 9，将制作好的 WeapomAmmo 控件蓝图添加到 PlayerInGameHUD 内，并调整其位置与尺寸。

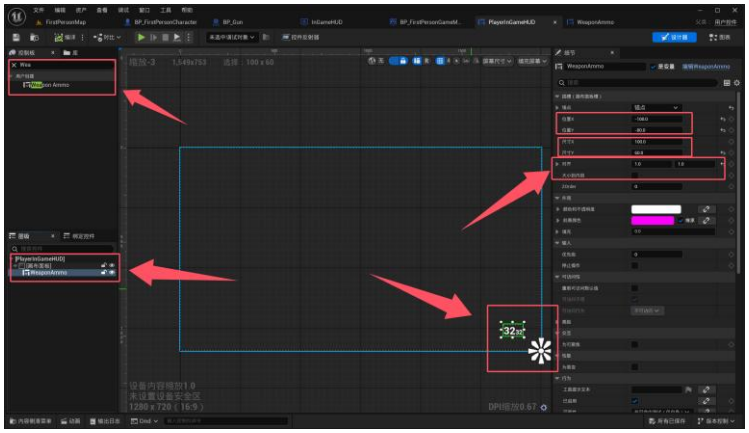

图 9 调整子弹数量空间蓝图在 PlayerInGameHUD 内的尺寸和位置

如图 10，在 BP_Gun 中添加两个变量：记录剩余子弹的 CurrentAmmo 以及

用于记录最大容量的 MaxAmmo。同时为了能够传递射击信息，需要再 BP_Gun 内添加事件分发器 OnShoot，用于传递发生射击动作。

为了方便记忆，将 BP_Gun 的名字修改为 BP_WeaponBase。

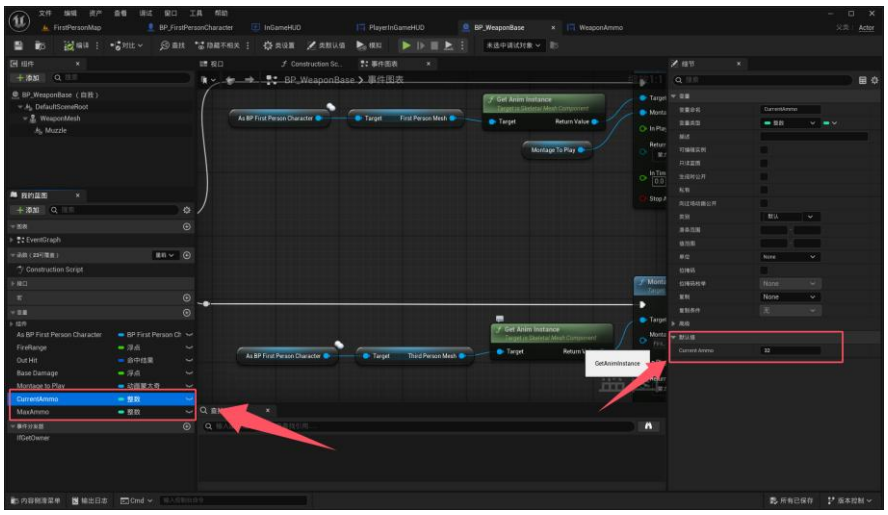

图 10 添加变量用于传递子弹数量

如图 11，在 WeaponAmmo 内添加一个 Weapon 变量用于确定获取那一把武器的弹药信息，使用 Weapon 实现获取武器信息的事件 FetchWeaponReference，

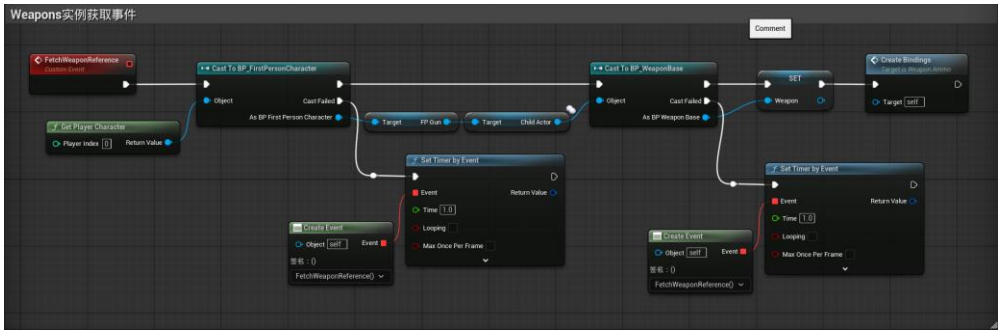

图 11 武器实例获取事件

在 FetchWeaponReference 内调用射击绑定事件 CreatBindings 其具体实现方式如图 12 所示：当实例获取成功则调用弹药更新事件 UpdateAmmo，否则重新获取武器实例。

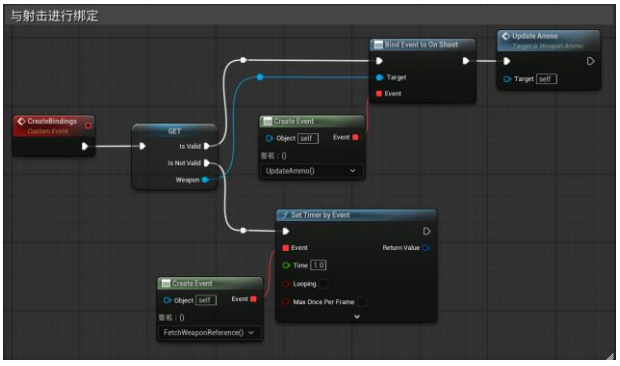

图 12 射击绑定事件

其中弹药更新事件如图 13 所示。

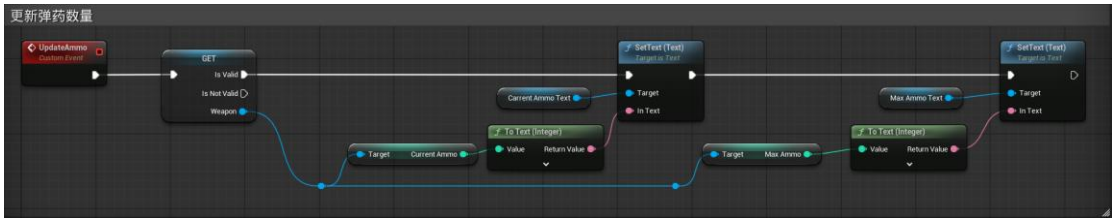

图 13 弹药更新事件

最后在事件预构造中调用 FetchWeaponReference

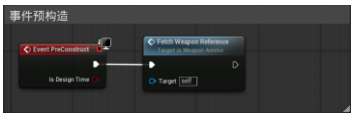

图 14 事件预构造

为了实现换弹后恢复子弹数量，这里对 BP_WeaponBase 内的 MCReload 事件进行如图 15 的修改：将 CurrentAmmo 的数值设置成 MaxAmmo，并通过事件分发器 OnReload 分发出去，同时更改蒙太奇动画的播放方式，即添加消息通知功能。在 WeaponAmmo 的 CreateBindings 内新增对消息的接收，并重新调用 UpdateAmmo 事件完成换弹操作（如图 16）。

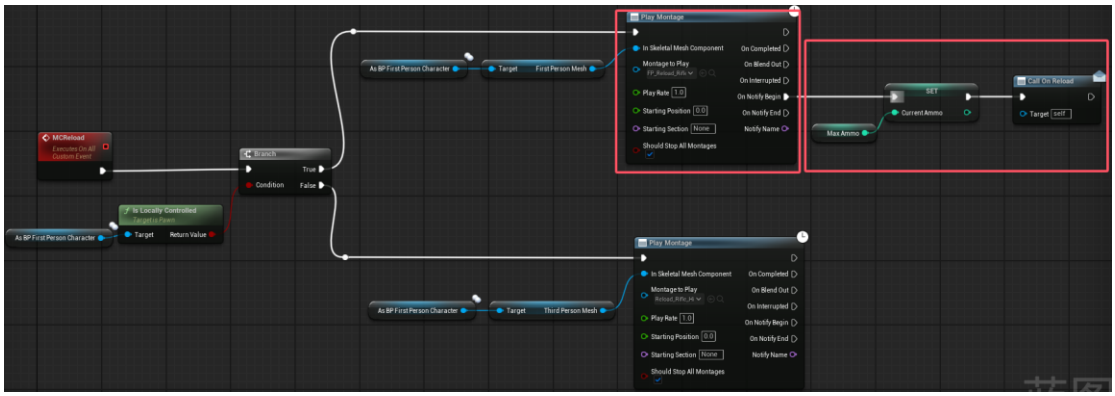

图 15 更新 MCReload 分发换弹信息

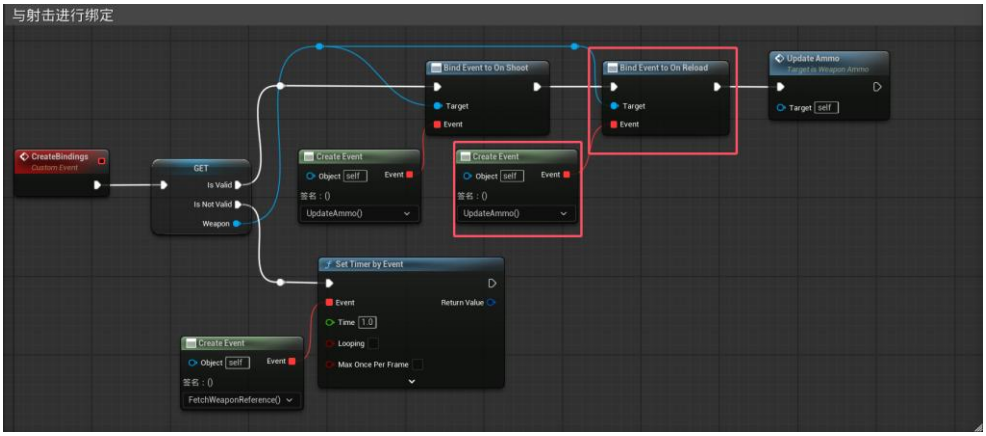

图 16 修改 CreatBindings 事件

如图 17，在动画蒙太奇 FP_Reload_Rifle_Hip_Montage 快结束时，在 1 那个轨道添加蒙太奇通知，用于通知更新弹药数量和更新 UI。

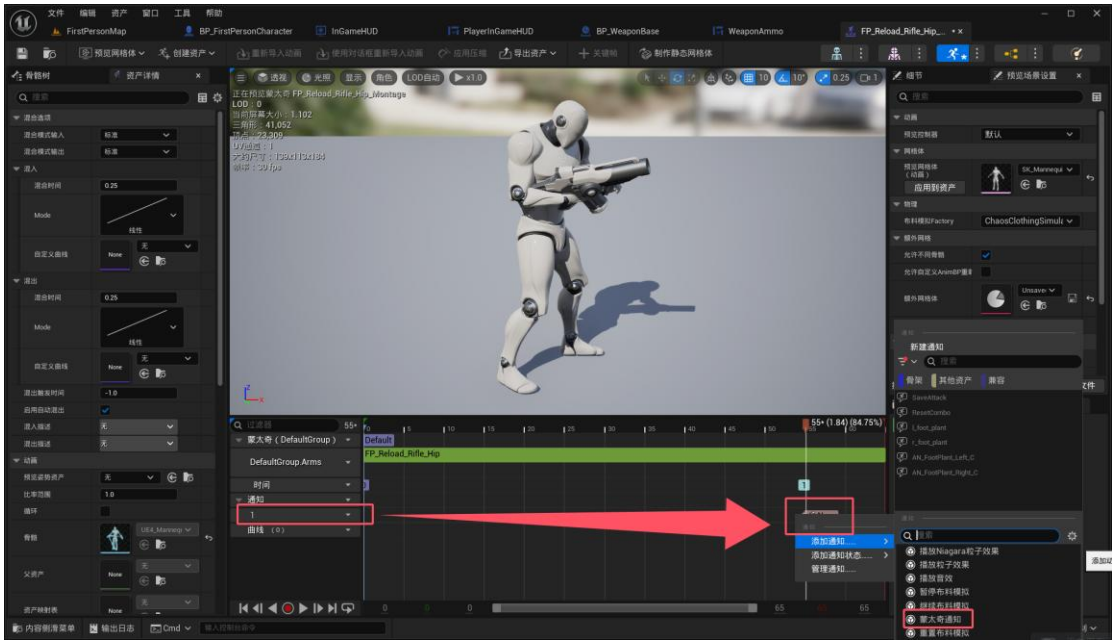

图 17 为蒙太奇动画添加通知

生命值 UI

在 BP_FirstPersonCharacter 内创建计算点状伤害的事件，计算血量变化，并通过事件分发器 OnDamage 通知 HealthBar 更新 UI。

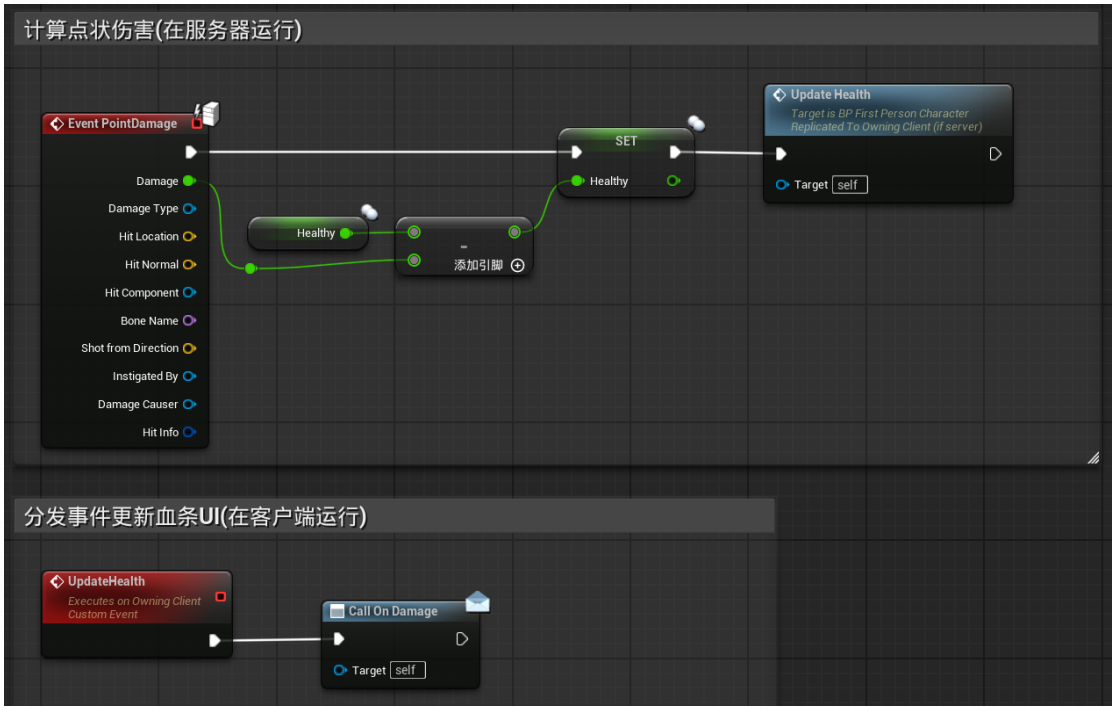

图 18 计算角色受到的点状伤害并通知更新 UI

在 GameUI 内新建一个控件蓝图，父类选择用户控件命名为 HealthBar。进

入用户控件，添加一个进度条命名为 ProgressBar，将其默认值设置为 1 且将其转变为变量（如图 19）。

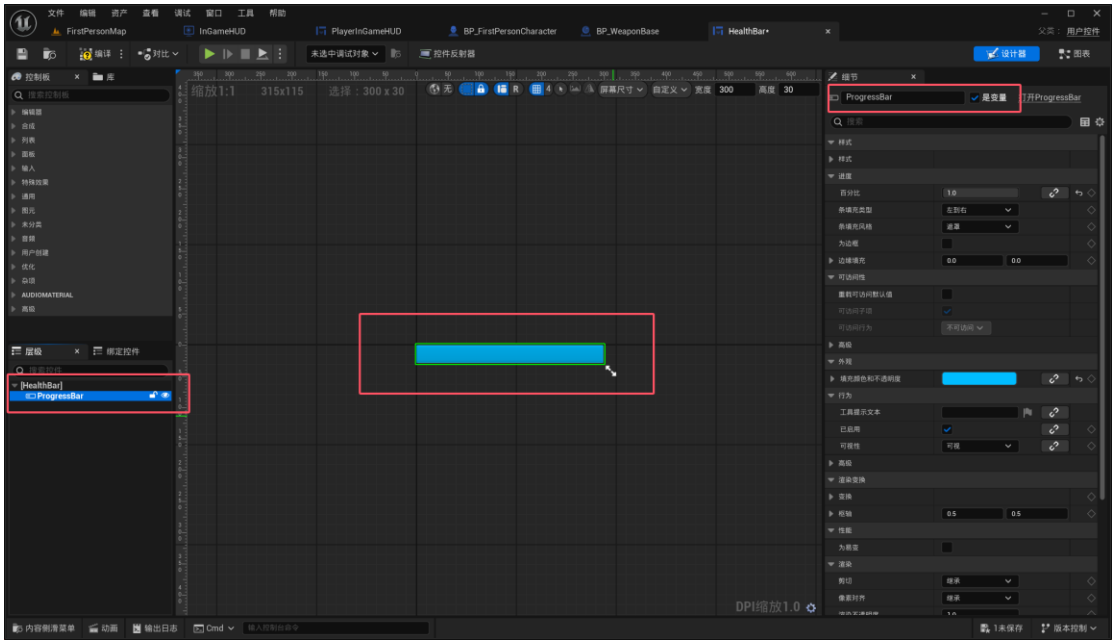

图 19 血条控件

如图 20 在事件图表中接收 OnDamage 分发的消息，并更新血条 UI 的百分比。

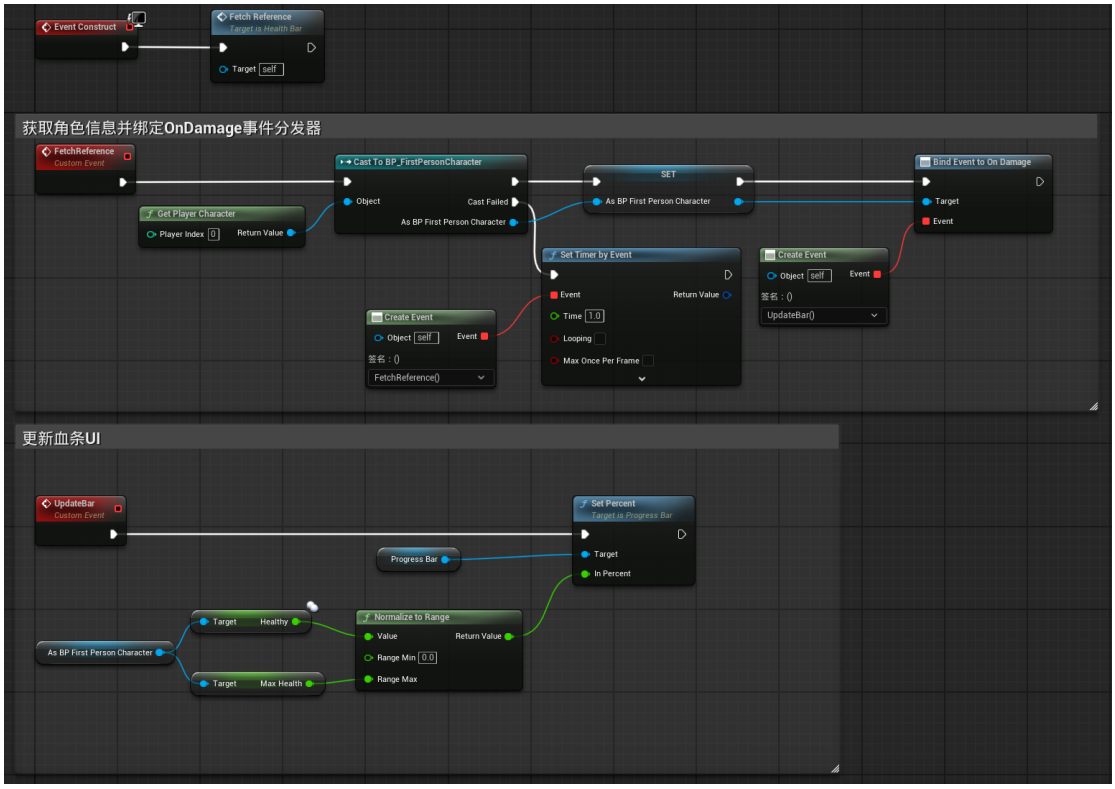

图 20 HealthBar 事件图表

如图 21，将创建好的血条控件 HealthBar 添加到 PlayerGameHUD 内。

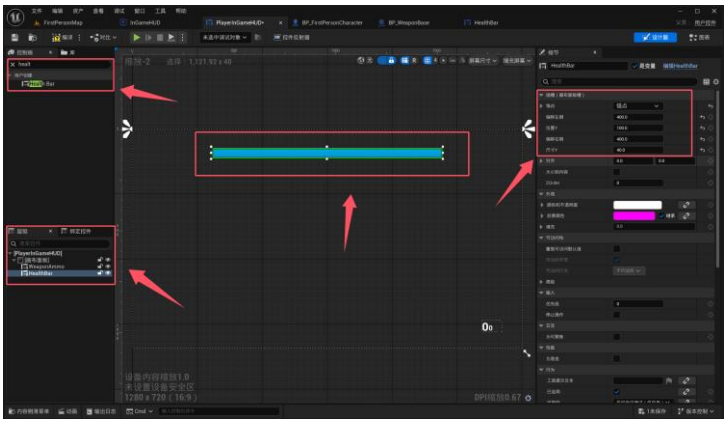

图 21 更新后的 PlayerInGameHUD

准星 UI

如图 22，在 UE 提供的免费资源中下载十字准星资产，将其导入项目。创建两个控件蓝图命名为 Reticle 和 ReticleChange，用于实现准星 UI 以及命中敌方后 UI 的改变。具体连接展示如下：https://d1iv7db44yhgxn.cloudfront.net/documentation/attachments/c8306563-314f-4abc-91fd-24835de32665/crosshair_fps_tutorial.zip）

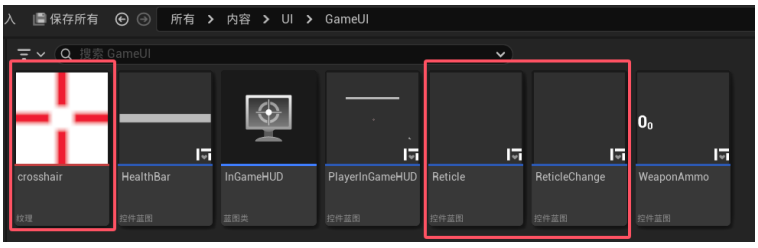

图 22 导入十字准星资产 crosshair 并创建控件蓝图

如图 23，在 Reticle 控件蓝图内添加 Image，并导入 crosshair，无需做任何修改。

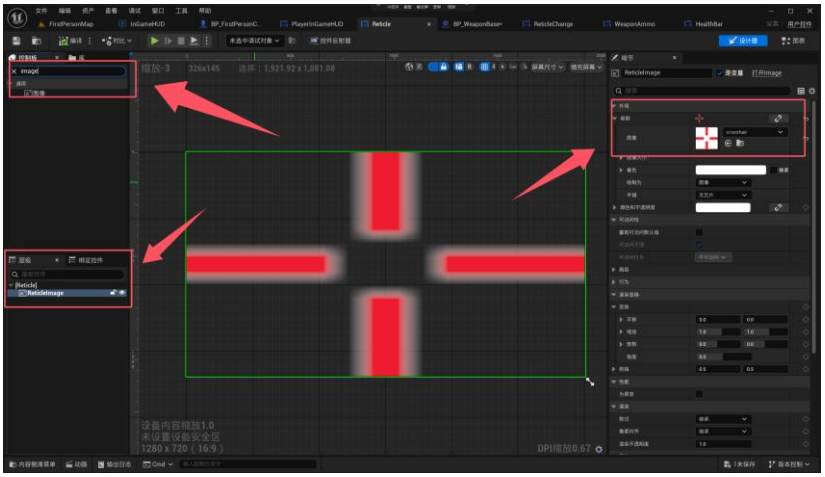

图 23 在 Reticle 内导入 crosshair

如图 24，以同样的方法向 ReticleChange 中导入 crosshair，但需要设置不透明度为 0，且旋转 45°。

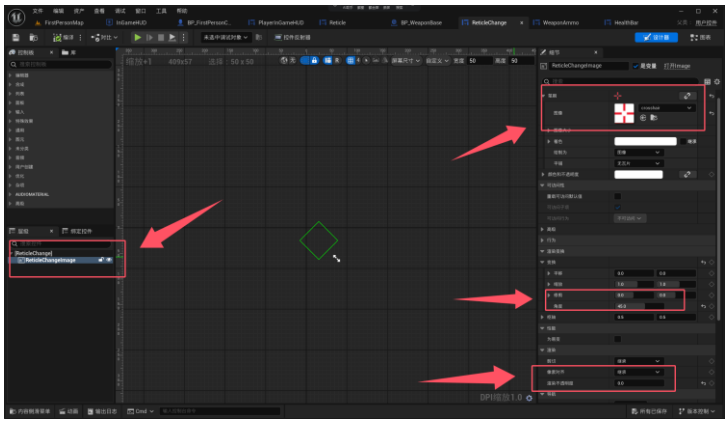

图 24 在 Reticle 内导入 crosshair

如图 25，为了能够获取武器是否命中敌方角色，需要在 BP_WeaponBase 内的事件 SeverShooting 的末尾添加判断命中敌方目标的功能，同时调用 UpdateHitInfo 传递命中信息，控制 ReticleChange 改变自身透明度。

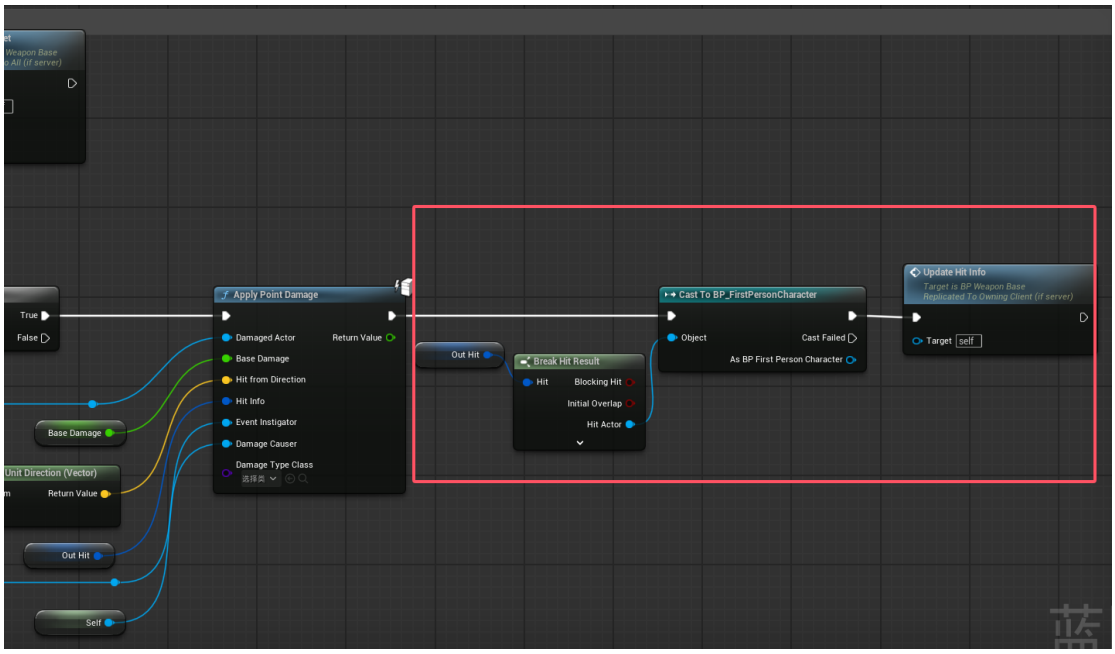

图 25 判断是否命中敌方角色

如图 26，UpdateHitInfo 需要将其复制属性设置为在拥有的客户端上运行。

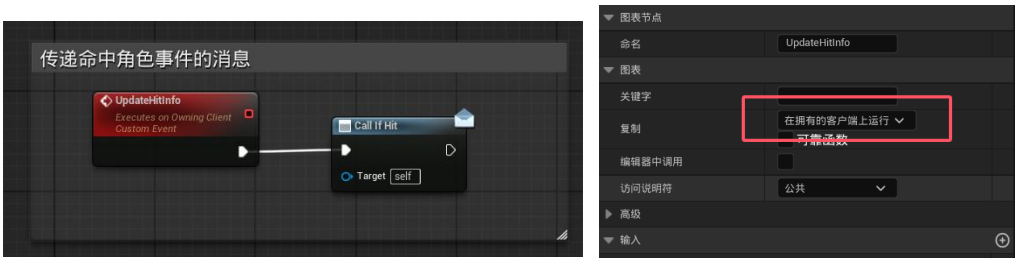

图 27 命中敌方消息发布

如图 28，在 ReticleChange 的事件图表内实现每一帧都调用 FetchWeaponm-Reference 获取当前角色的武器实例，判断其是否命中敌方目标。如果命中则调

用 UpdateOpacity 改变准星图像。

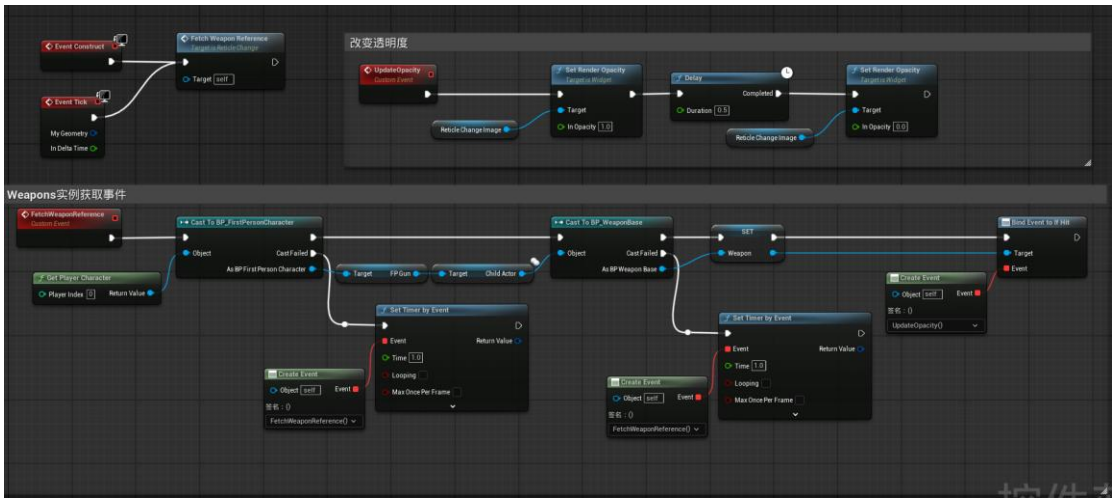

图 28 ReticleChange 的事件图表

具体实现效果展示如图 29：当命中对方角色时，准星会变为米字形，却敌方角色会损失血量，且右下角会有子弹数量的衰减。

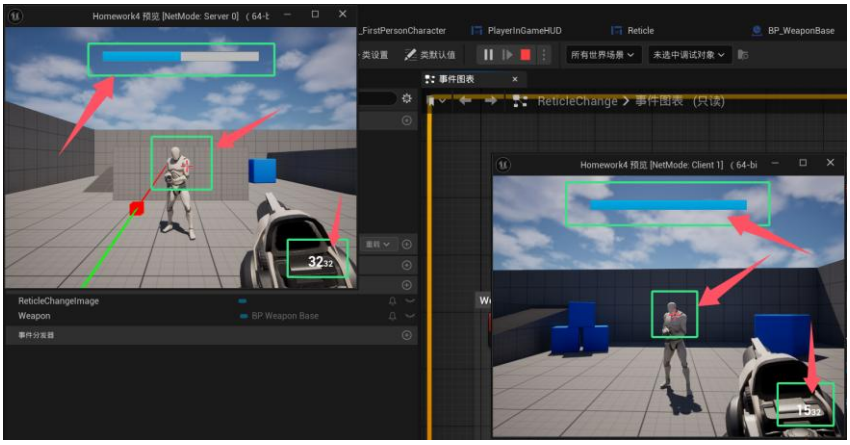

图 29 实现射击游戏操作面板

登录界面与加载进度条

在 FirstPerson/Maps 内创建一个空关卡，命名为 StartMap 如图 30。

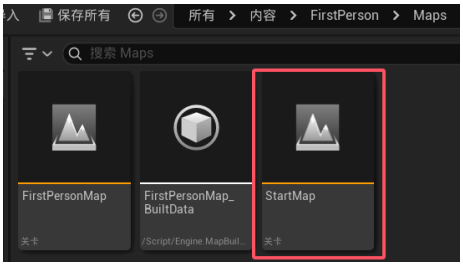

图 30 创建空关卡

在“编辑”内找到找到项目设置，选择地图与模式，将编辑器开始地图和游戏默认地图均选为 StartMap，如图 31。

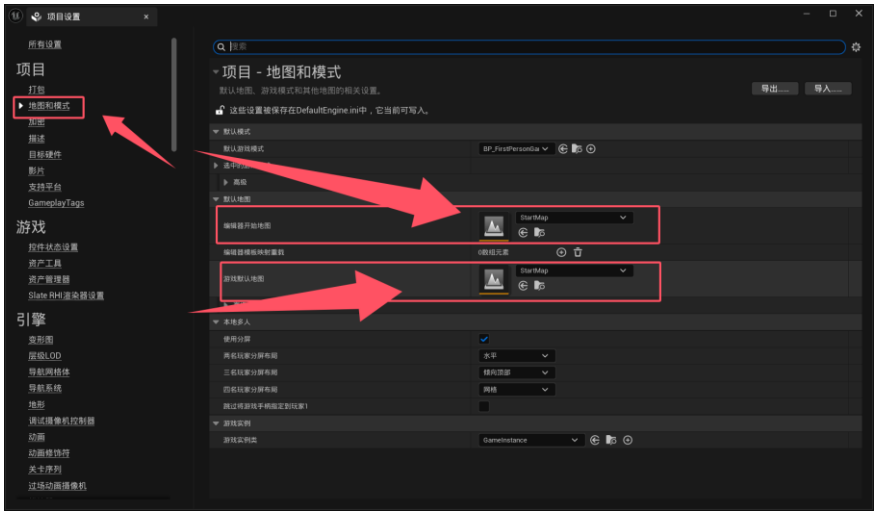

图 31 修改初始地图为 UI 关卡

基础登录 UI 设计

制作 UI 界面的树形结构如图 32 所示， 其中画布面板用于展示 UI 背景，边界用于控制范围，register 是注册相关界面，使用垂直框来堆叠各个组件，文本主要用于传达文本提示 RUsername、RPassword 以及 RCPassword 分别是注册用户名、注册密码和注册确认密码的文本框。Button_365 是注册，Button_1202 数退出，用于返回登录界面。LUsername 和 LPassword 分别是登录界面的用户名和密码，Button_1 用于进入注册界面，Button 用于实现登录。Button_1577 用于退出游戏，尺寸框用于包含进度条 ProgressBar_149，用于实现游戏加载动画。注：所有与 Password 有关的文本框均设置为“为密码”。

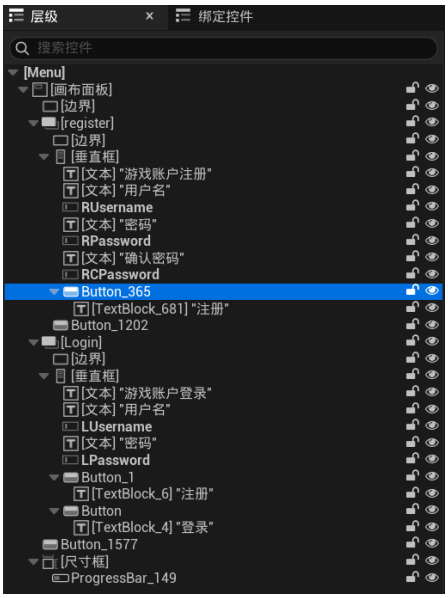

图 32 UI 的树形结构

各情况的 UI 界面展示在图 33 内，其中第一幅图为注册界面，第二幅图为登录界面。进度条的隐藏与加载，以及用户的注册与登录逻辑将在后续进行展示。

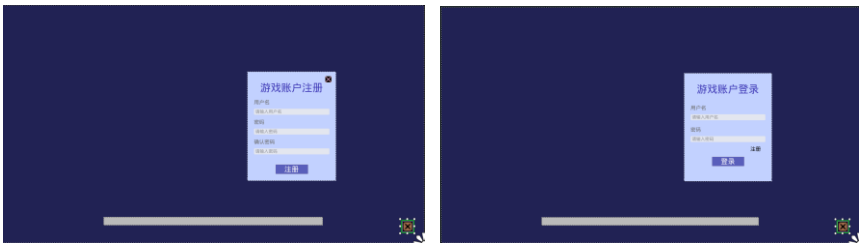

图 33 UI 展示

其中，对于文本类的设计主要是调整大小并修改颜色。登录以及注册按钮的设计是修改背景颜色，并为其添加文本描述。各种输入文本的文本框则是修改颜色并调整提示文本。注册退出按钮选择了纹理 Bad，游戏退出按钮选择了纹理 BlendFunc_DefBlend，具体的数值设计不在此处展示。

在 StartMap 关卡蓝图中将 UI 显示出来，具体的蓝图设计如图 34 所示。

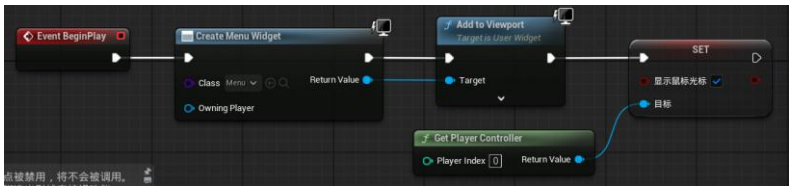

图 34 在 StartMap 内显示 UI 的蓝图设计

页面转换的功能实现

此部分主要实现点击登录界面的注册按钮后，跳转到注册页面；点击注册页面的退出按钮后，返回登录页面；点击右下角的退出按钮时退出游戏。

退出按钮

在设计器中选中右下角的退出按钮，跳转到图表界面。如图 35，添加“点击时事件”。为该事件添加退出游戏功能如图 36 所示。

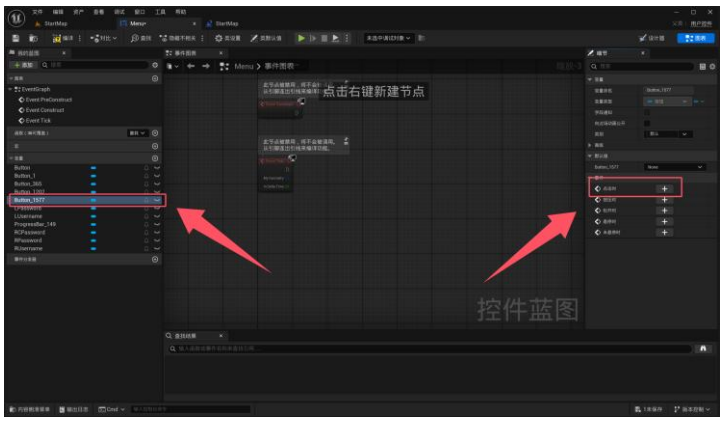

图 35 添加退出按钮点击时事件

图 36 退出游戏点击时事件蓝图

登录界面的注册按钮

将登录界面注册按钮中的文本设置为变量“TextBlock_6”，并将登录界面的注册按钮 Button_1 的“按压时”、“松开时、“悬停时”和“未悬停时”添加到事件图表内，如图 37.

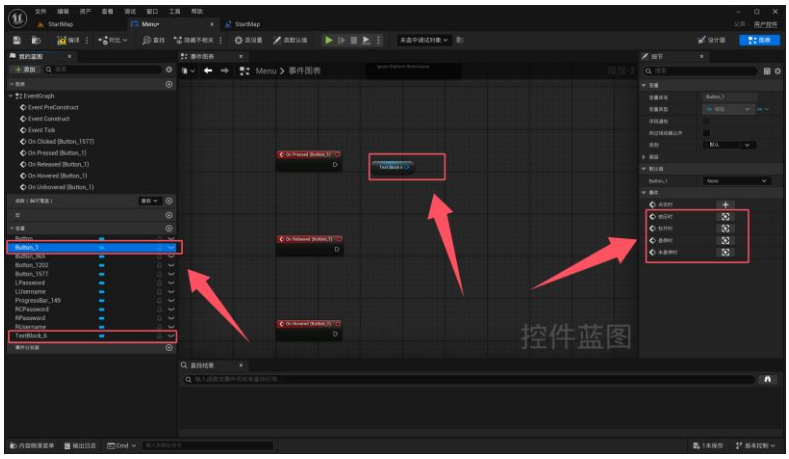

图 37 添加登录界面注册按钮相关事件蓝图

按钮颜色变化，如图 38 以“按压时”的事件设计为例，在按压后将“注册”两个字的颜色修改为想要转变的颜色。其余状态的颜色转变逻辑于此相同，进需要修改 SpecifiedColor 的颜色即可。

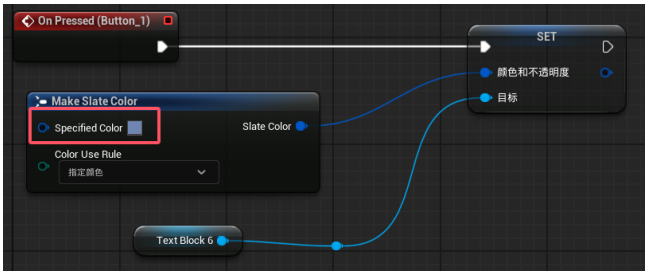

图 38 按压时按钮颜色转变

为了能够实现注册页面的跳转，需要先将注册页面隐藏，并在点击注册按钮后，隐藏登录页面并显示注册页面。

设计一个动画命名为“RegisterAnim”，先将注册界面添加进来，如图 39。登录界面以同样的方法添加到动画 RegisterAnim，用于后续界面转化动画的制作。

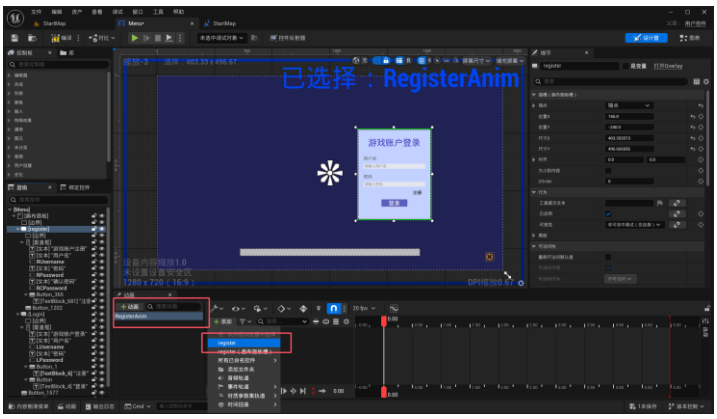

图 39 创建动画并添加注册页面

如图 40，分别为二者添加变换和渲染不透明度的轨道。在时间轴为 0 时，将注册界面的转换轨道内将缩放调整为 0，并将不透明度也调整为 0。

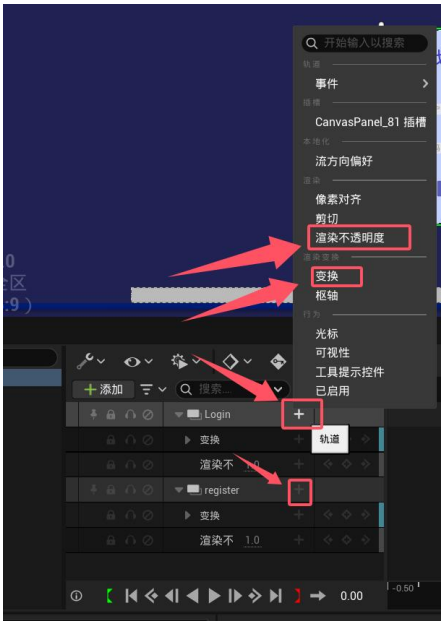

图 40 为界面添加动画轨道

如图 41，在登录界面注册按钮按压时的事件蓝图内添加播放动画的功能，实现登录界面到注册界面的转换。

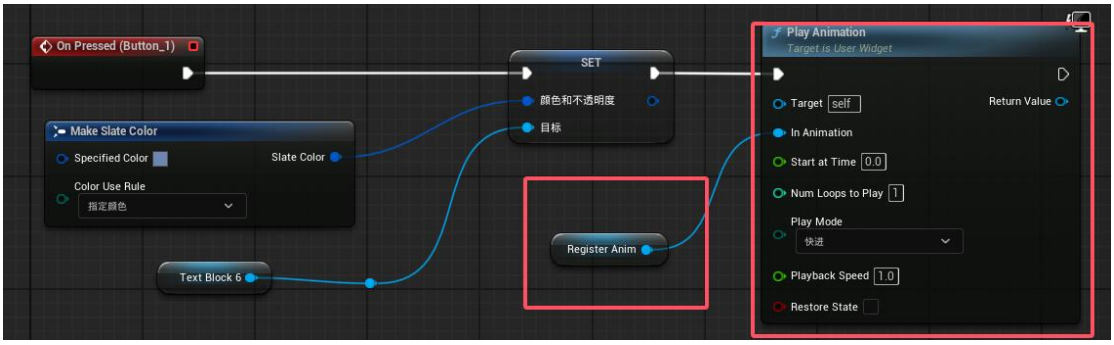

图 41 添加登录界面到注册界面转换的动画

注册界面的返回按钮

将返回按钮 Button_1202 的点击时事件添加到事件蓝图，并将图 40 设计的动画翻转播放，具体修改如图 42 所示。

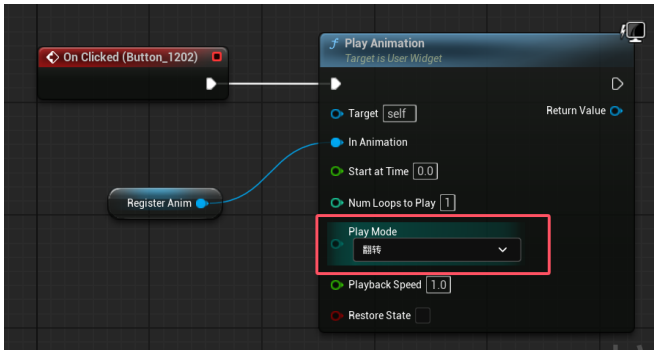

图 42 注册界面返回按钮的事件蓝图

注册功能的实现

在基础 UI 的基础上，添加文本组件命名为“Remind”用于提示用户操作的错误信息，将其调整到合适的位置，并把渲染不透明度调整为 0。在事件图表中创建文本变量 RemindText，并将 Remind 的文本信息绑定到 RemindText。

图 43 绑定文本并设置提示文本格式

如图 44，为提示文本制作动画 RemindAnim，将 Renind 文本块添加到动画内，并添加变换和渲染不透明度的轨道。在时间轴为 0 时修改 y 轴平移为 25，在时间轴为 0.25 时，修改 y 轴平移为 0，渲染不透明度为 1。在时间轴为 0.75 秒，为平移和渲染添加关键帧。

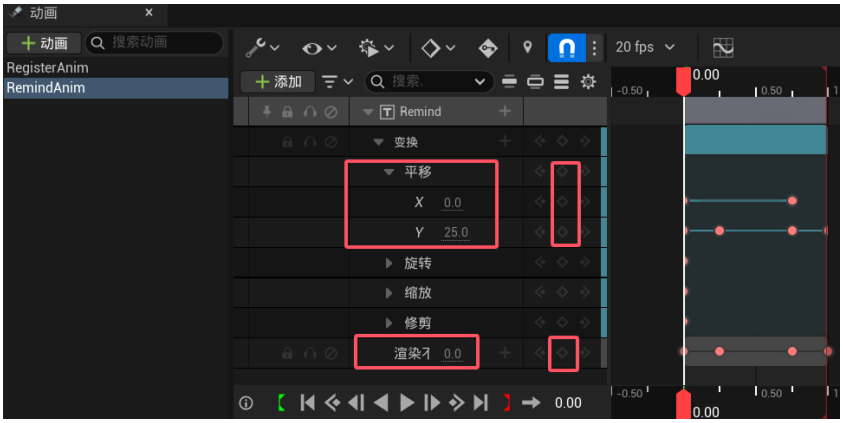

图 44 提示文本动画

如图 45，创建播放动画的函数 RemindFunction，并为其添加类型为文本的输入，用于将值传递给 RemindText 变量，并播放动画 RemindAnim。

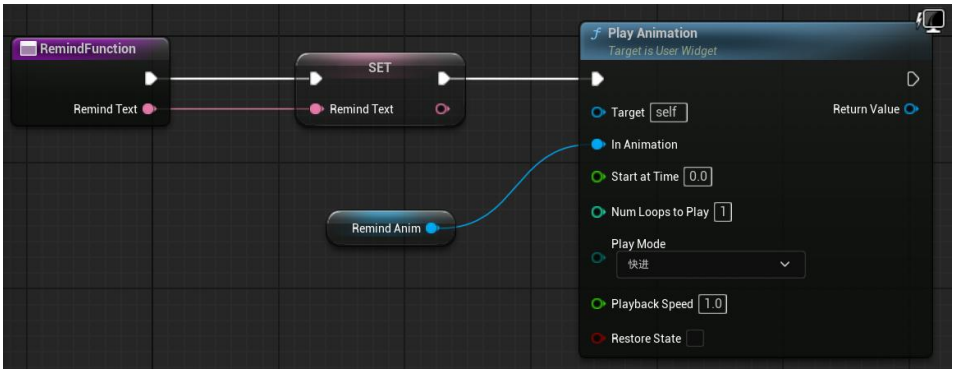

图 45 提醒文本动画蓝图

如图 46，将 RUsername、RPassword 以及 RCPassword 的文本提交时事件添加到事件图表，主要用于获取用户的相关输入，并判断用户此次输入是否符合要求。创建 RegisterUsername、RegisterPassword 和 RConfirmPassword 三个文本变量用于保存用户输入的用户名、密码和确认密码。创建文本数组 AllUsername 和 AllPassword 用于保存全部已注册成功的用户名和密码。

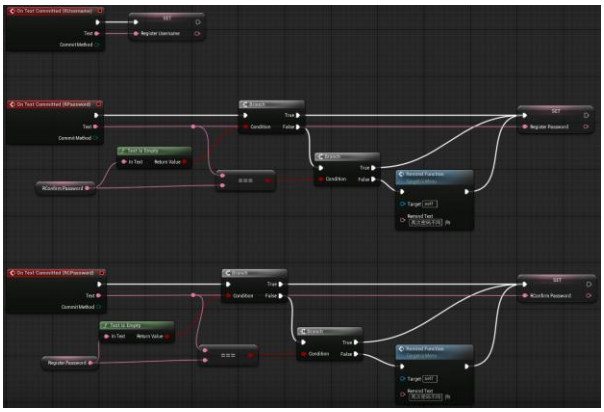

图 46 保存用户输入并判断是否合法

如图 47，将注册界面的注册按钮的点击时事件添加到事件图表，首先判断注册用户是否为空，如果为空则提醒“用户名为空”，然后判断两个密码的状态，如果有一个及以上为空或二者不相同，提醒“密码错误”。上述判断均通过后，将用户输入的用户名及密码分别添加到 AllUsername 和 AllPassword 内，并提示用户注册成功。

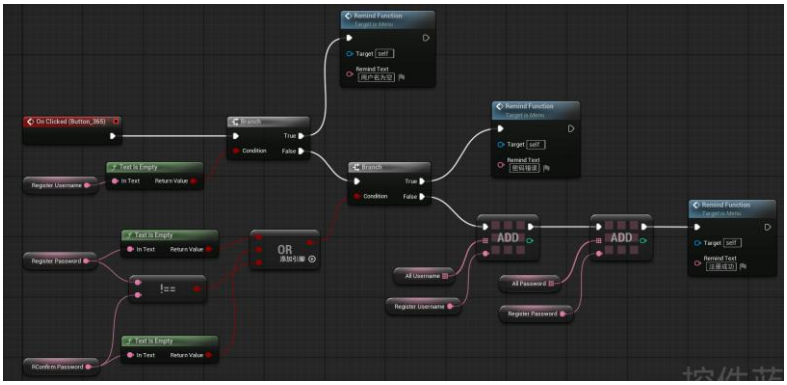

图 47 判断注册信息是否合法并完成注册内容的添加

登录及加载功能的实现

登录逻辑

如图 48，将 LUsername 和 LPassword 文本框的文本提交时事件加入事件图表，并新建 LoginUsername 和 LoginPassword 两个变量用于保存用户输入的用户名和密码。

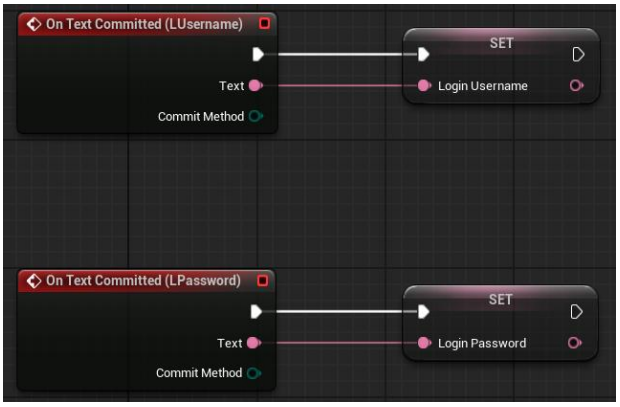

图 48 获取用户登录时输入的用户名和密码

如图 49，为登录功能的实现，需要新建一个变量 HasUsername 用于保存是否能在数组 AllUsername 内找到对应用户名。具体实现实现流程为，先引入循环遍历整个数组，并判断是否有用户名与用户输入的用户名相同，如果相同将 HasUsername 设置为真并打断循环，否则将 HasUsername 设置为假。循环结束后开始判断登录情况，分别为 HasUsername 为假时输出“用户名错误”，为真时判断对应密码是否相同，密码不同则输出“密码错误”，否则执行登录成功的后续步骤。注：这里在登录成功后需要将 UI 界面设置为不可操作。

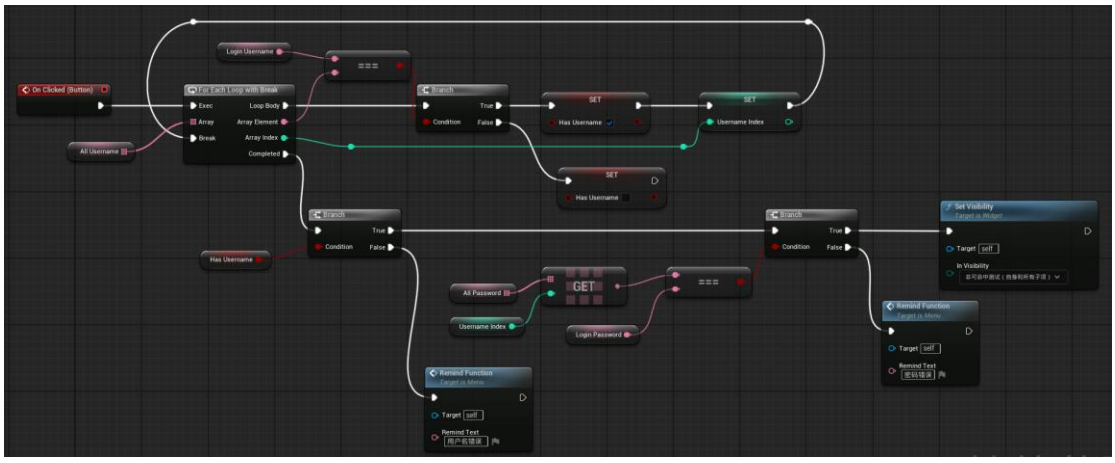

图 49 登录逻辑的事件蓝图

进度条与登录后操作

将进度条的可视性设置为易折叠，使得进度条在默认状态为不可视。编写

Loading 事件如图 50，用于实现进度条的加载动画，即按照时间加载进度条，同时当其大于 1 时跳转关卡。并在图 49 的 SetVisibility 后添加按照函数名执行的计时器，并在计时器后面将进度条显示出来。

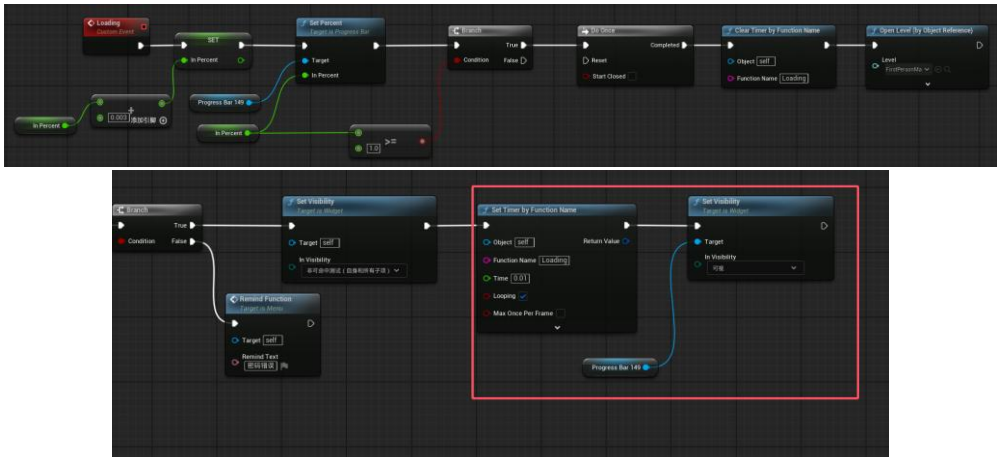

图 50 Loading 事件以及登录后需执行操作

注册数据存入本地

新建一个蓝图类，继承 SaveGame，在其中创建两个文本类型的数组变量命名为 AllUsername 和 AllPassword。修改图 47 的事件图表如图 48。在用户输入数据合法时，判断是否有时间插槽 LoginInfo 存在，如果不存在则创建对应的事件并调用 SaveLocal 函数；如果 LoginInfo 存在，判断用户要注册的信息是否和 AllUsername 重复，如果重复则结束循环，提示“用户名已存在”。如果输入信息与 AllUsername 不重复，则获取到 BP_SaveGame，修改其中内容。

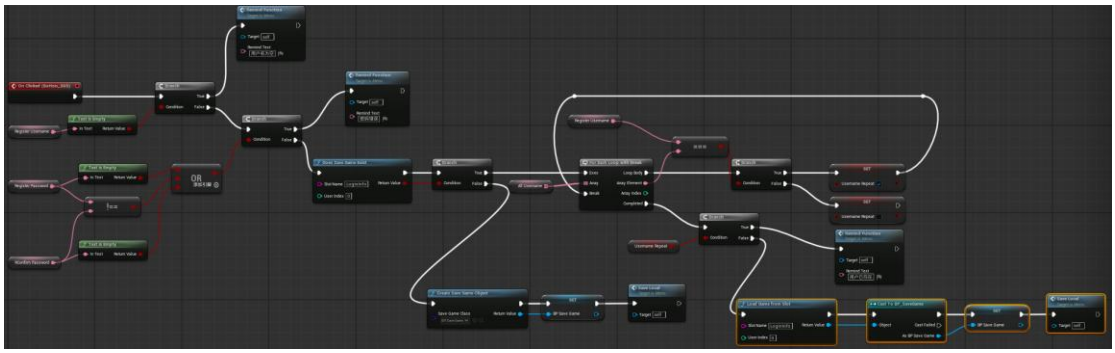

图 48 为注册添加保存本地的功能

其中 SaveLocal 的函数内容如图 49 所示，将用户输入的信息加入数组，并将数组传输给 BP_SaveGame 的相关数组内，将数据保存到本地，保存成功后情况注册 UI 内的原有输入文字，方便用户继续注册用户名。

图 49 SaveLocal 函数

最后在游戏开始运行时，如果本地已有 LoginInfo 的相关信息，则将其已有信息

直接传输到 AllUsername 和 AllPassword 内，初始化全局数组。

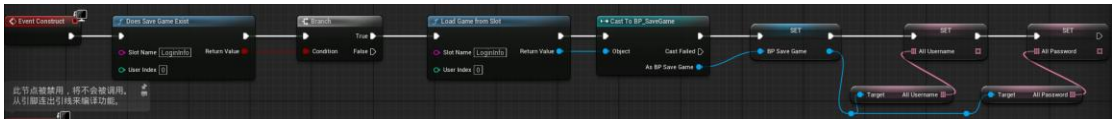

图 50 初始化全局数组

具体的实现效果详见 Homework4 的展示视频。

Homework4.1：在游戏基础 UI 部分只做了角色弹药数，命中玩家后的准星反馈，血条显示的功能，同时当弹药打光之后按设计也无法进行发射。

Homework4.2：登录界面 UI 包括判断用户名是否正确，判断密码是否正确，注册文本的填写是否合法，注册后的信息已保存本地（关闭执行后仍能够使用已注册的用户进行登录），以及退出登录界面，登录后能够显示进度条。