前言：

本作品使用 unity 制作，且使用了 unity 中的通用渲染管线（URP），但除此之外并无使用其他的包。本作品的系统设计部分参考了游戏《星际争霸 2》中用于制作地图的《银河编辑器》，设计可能会出现相似之处。

本文档将介绍本作品中的重要系统的设计与实现方式，这些系统包括：自定义资源、效果系统与部分重要效果、过滤器、通用属性集与其子类、通用控制器、单位事件与事件扳机、技能系统、武器系统、通用 AI、状态系统、剧情系统。

对于这些系统，本文档将从系统介绍与设计目的、主要实现过程来讲解，部分系统还会有举例的过程。

正文：

一、 自定义资源

系统介绍与设计目的：

此系统是本作品的核心系统之一，在作品中被大量的使用。此系统的设计目的有两个。

第一个目的是为了将所有可能被修改的资源文件外置，以方便调试与修改。这些资源包括但不限于武器、弹药、效果、剧情、准星等。

第二个目的是方便游戏日后的 mod 功能。外置的资源文件可以让玩家很方便的增加与修改，能极大地方便玩家开发游戏 mod。

主要实现过程：

本系统中的所有资源类均直接或者间接的继承自 unity 的 ScriptableObject，类，此类能够提供实例变量的序列化与反序列化功能，同时也能通过重写 Editor 来实现自定义的资源编辑器。

所有的自定义资源类均提供方法获取资源的实例，在调用此方法时，会生成一个当前资源的实例副本并返回。

二、 效果系统与部分重要效果

系统介绍与设计目的：

此系统同样时本作品的核心系统之一，同时，所有的效果均是自定义资源。本作品对于效果的定义是：游戏内单位与其他单位、游戏本身通信的唯一方式。通信内容包括但不限于攻击、修正、添加状态等。

此系统设计的目的主要目的是保证游戏内单位的一切交互均是可捕捉的，可拦截的，因为所有的通信均要经由效果系统。

同时，通过效果间的组合与连接，可以实现所有的技能效果，使游戏的后续开发过程基本脱离代码编写。例如，在游戏中，Boss 小雪的“雪落斩击”会在玩家周围产生多个刀光，在小雪收刀时刀光爆炸造成伤害。此过程主要触发了以下效果（并非所有触发的效果）：

等待动画事件→
延迟 1（9 段短延迟，共触发 9 次下效果）→
随机点（在玩家周围随机选择数个点，此处为 1 个，并随机选取方向）→
创建游戏对象（创建刀光，刀光在触发者上注册为召唤物）

等待动画事件→
召唤物通信（与自身召唤的刀光通信，引爆刀光）

效果被分为三类效果：
基础效果：包含伤害、添加状态标签、添加属性修正、推动等效果
逻辑效果：包含复合、对象变换、运算填充、角度偏移等效果
全局效果：目前仅包含 停顿与抖动

主要实现过程：

所有的效果均继承 Effect 基类。

Effect 类拥有两个属性：目标类型与是否接受方向向量。目标类型分为游戏对象、点、全部。Effect 类定义了四个需要子类实现的方法，分别是对游戏对象、点和包含与不包含方向向量的生效函数。（代码如下）

```
public class Effect
{
    40 个引用
    public enum EffectType
    {
        Point,
        GameObject,
        Both
    }
    public bool DirectionAllow = false;
    public EffectType Type = EffectType.Both;
    3 个引用
    public virtual void Fill(string variableName, float value) { }
    24 个引用
    protected virtual void takeEffect(GameObject trigger, GameObject target) { }
    17 个引用
    protected virtual void takeEffect(GameObject trigger, Vector2 target) { }
    17 个引用
    protected virtual void takeEffect(GameObject trigger, GameObject target, Vector2 direction) { }
    15 个引用
    protected virtual void takeEffect(GameObject trigger, Vector2 target, Vector2 direction) { }
    30 个引用
    public void TakeEffect(GameObject trigger, GameObject target)...
    13 个引用
    public void TakeEffect(GameObject trigger, Vector2 target)...
    21 个引用
    public void TakeEffect(GameObject trigger, Vector2 target, Vector2 direction)...
    37 个引用
    public void TakeEffect(GameObject trigger, GameObject target, Vector2 direction)...
}
```

图 1 Effect 类的代码

在触发生效函数时，需要传入触发者与目标，也可以同时传入方向向量。触发者与目标二者只要有一个不存在（即被游戏销毁了），效果就不会被触发

值得注意的是，如果目标类型不包含点，但是触发了对点的生效函数，那么此次触发将被忽略，游戏对象同理。但如果选择了不接受方向向量，但传入了方向向量，那么效果会正常生效，但会忽略方向向量。

效果举例：复合

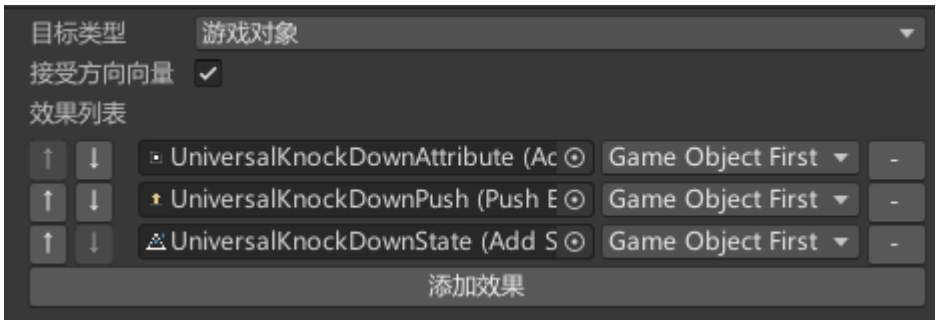

图 2 重写的复合效果的编辑器界面（图示为部分的小雪被击倒效果）

此效果在游戏运行中创建时，会生成一个复合效果的实例，并传入效果列表的所有效果的实例。在触发时，会触发效果列表中的所有效果的对应的方法。例如，触发 TakeEffect(GameObject, GameObject)时，会触发效果列表中的所有效果的 TakeEffect(GameObject, GameObject)。

效果举例：运算填充

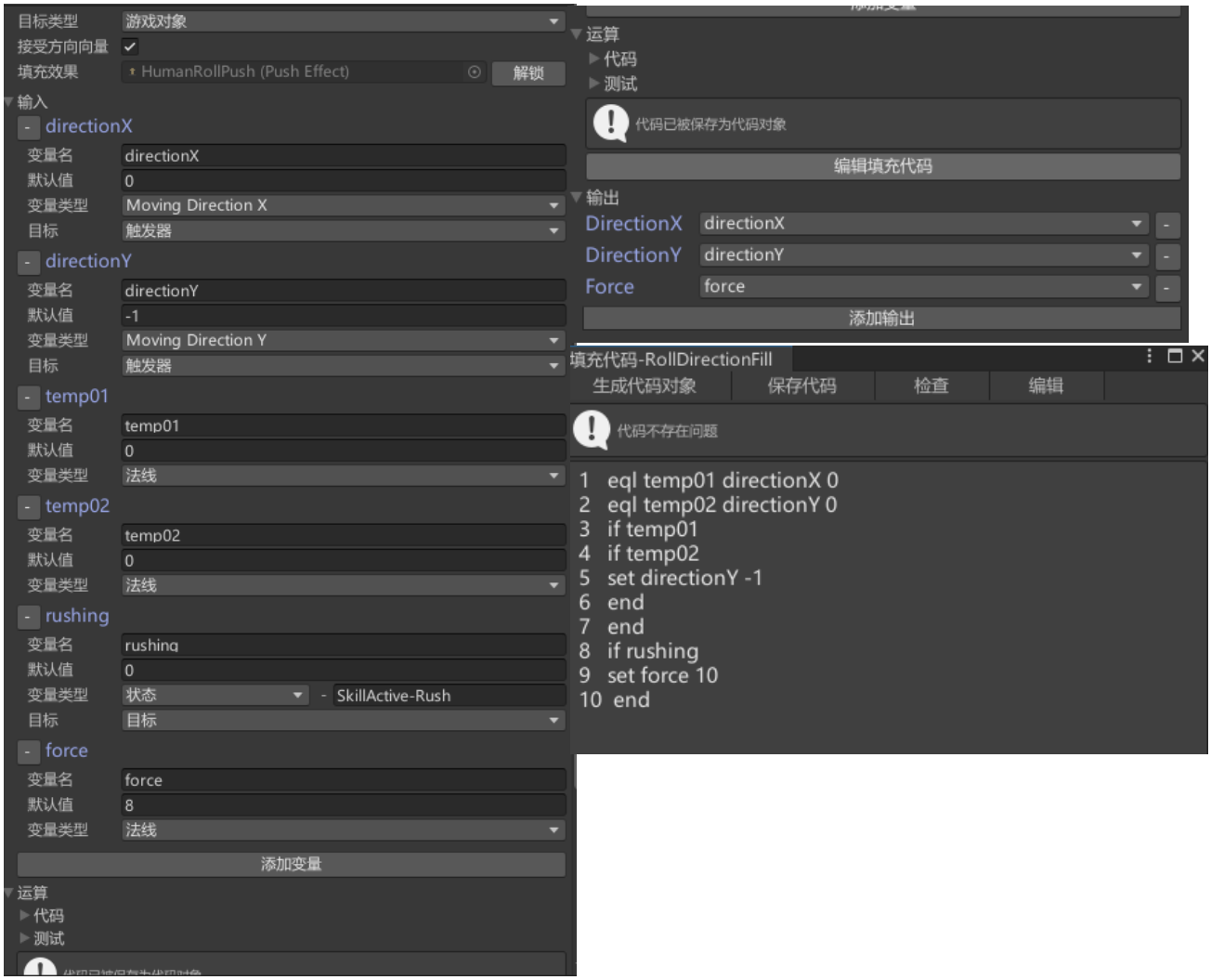

图 3 4 5 重写的运算填充效果的编辑器界面（图示为翻滚的部分效果）

此效果用于在运行中动态地修改其他效果或方向向量。此效果从全局变量、目标、触发者处获取输入，存入变量中，并使用我模仿汇编语言创造的 Fill 效果语言进行运算。最后将结果输出至其他的效果中。

在触发此效果时，此效果会进行运算，修改下一效果内指定变量的值，然后触发下一效果。在上述的截图中，该效果检测了目标是否处于冲刺状态中，如果在冲刺状态中，则修改下一效果（推动）的力度。截图中的效果是为了让人物在冲刺状态下能够翻滚的更远，且修改默认方向向量向下。

Fill 语言支持四则运算、随机、取整、判断、循环功能，但仅支持 float 类型的变量，且所有变量必须在运算前被定义。

三、 过滤器

系统介绍与设计目的：

过滤器也是一种自定义的资源类型，过滤器是为了筛选效果、技能、武器的目标。

过滤器分为以下两类：

基础过滤器：常量过滤器、状态标签过滤器、队伍过滤器、速度过滤器等

逻辑过滤器：复合过滤器和否定过滤器

主要实现过程：

所有过滤器都继承自 Filter 类，Filter 类拥有供子类重写的拥有触发者与不拥有触发者的过滤方法。

```
80 个引用
public class Filter
{
    19 个引用
    public virtual bool Judge(GameObject trigger, GameObject gameObject)
    {
        return Judge(gameObject);
    }
    16 个引用
    public virtual bool Judge(GameObject gameObject)
    {
        return true;
    }
}
```

图 5 Filter 类的代码

四、 通用属性集与其子集

系统介绍与设计目的：

此系统是单位拥有的构件（Component），属性集中的属性会被序列化保存。通用属性集的属性包含最大生命、初始生命、最大法力值、初始法力值、最大体力值、初始体力值、护甲、抗性、受击率等，子集会包含额外的属性，如射手属性集包含精度加成、瞄准速度加成等。

所有在属性集中的属性均支持修正，修正会暂时的修改属性的值，而不改变其初始值。修正分为值修正与比率修正。修正对于尝试读取属性集内属性的其他类是不可知的，他们仅能获取修正后的属性。

属性集还负责管理状态标签与单位标签，状态标签是临时存在的，例如，技

能激活，不可移动等。单位标签是固有的，游戏运行后便不可修改。

此系统的设计目的是为了方便与规范化单位属性的管理。

主要实现过程：

属性集内拥有修正属性字典与修正数量字典，当一个属性被读取时，如果他在修正数量字典内的修正数量为 0，那么将直接返回初始值，否则，计算修正后返回。

属性集每 0.02s 会检测一次修正与状态标签持续时间，并移除超时的标签与修正。

举例：

角色的冲刺就是通过对最大速度属性添加修正实现。角色的翻滚为了延长距离，也会对加速度进行修正。

图 6 重写的通用属性集编辑器界面（图示为小雪的通用属性集）

五、 通用控制器

系统介绍与设计目的：

通用控制器是用于控制角色移动行为的。所有的移动均基于对于 unity 内刚体的速度修改，而在本游戏中，所有的速度修改调整行为均由通用控制器完成，其他类、系统想要控制单位移动均需要通过通用控制器执行。

通用控制器依赖于通用属性集或其子集，因为通用控制器会读取最大速度与加速度。

通用控制器对外开放 Direction 属性（二维向量），通用控制器会尝试将速度调整到 Direction 的方向上，并加速至最大速度。

通用控制器中还负责单位的转向行为，但也同样定义了一种控制器状态——当单位拥有“LostControl”状态标签或者当前速度超过最大速度时，通用控制器会尝试停止移动，而不是朝 Direction 方向移动，同时，此状态下单位转向将被禁用。

主要实现过程：

每 0.02s，通用控制器会检测单位状态，如果单位可控制，则会将当前速度加上加速度乘 Direction，如果速度的模大于最大速度且小于最大速度+加速度，则重新将速度的模设为最大速度，最后赋值给刚体的速度。

六、 单位事件与事件扳机

系统介绍与设计目的：

单位事件是单位在触发某些条件是会触发的事件，这些事件拥有不同的参数，但都可以被事件扳机捕获，事件扳机可以选择拦截事件，使本次事件不会造成结果。

例如，当玩家拥有护盾且被击中时，会触发两个事件。首先触发的是“被击中”扳机，这个事件会被护盾的事件扳机拦截，阻止默认的血液飞溅特效，并产生一个护盾的被击中效果（通过 创建游戏对象效果 实现）。第二个触发的事件是受伤，这个事件同样会被护盾的事件扳机拦截，阻止默认的扣除血量，并扣除对应的护盾值。

单位由一些默认的事件扳机，目前主要负责单位的韧性削减后的被击退与被击飞等基本所有单位通用的扳机。

单位事件与单位扳机的设计还主要为了后文将要提到的状态系统服务，具体内容见后文。

主要实现过程：

伤害效果、子弹命中等并非直接访问并修改单位的通用属性集的属性，而是访问单位的事件管理器，事件管理器会循环访问拥有的事件扳机，当没有事件扳机拦截此事件时，才允许此事件触发。

所有的事件扳机都继承自 EventTrigger 类，子类可以通过重写 EventTrigger 类中的函数来捕获并拦截事件。事件扳机还有用优先级与是否在拦截后触发两种选项。事件扳机在触发时会按照优先级触发；如果选择了在拦截后触发，那么即使事件已经被先触发的扳机拦截，依然会触发此扳机的对应方法。

```
public class EventTrigger
{
    public bool IsDestory = false;
    public bool AllowRepeat = true;
    public int Priority = 1;
    public GameObject AttachObject;
    // return whether break the event or not
    2 个引用
    public virtual bool Die()
    {
        return false;
    }
    2 个引用
    public virtual bool HurtPhysical(GameObject trigger, float value, Vector2 direction)
    {
        return false;
    }
    2 个引用
    public virtual bool HurtMagical(GameObject trigger, float value, Vector2 direction)
    {
```

图 7 EventTrigger 类的部分代码

七、 技能系统

系统介绍与设计目的：

技能系统是一种自定义资源系统，技能是一种自定义资源，可以添加到单位的技能管理器中。

技能系统中将技能划分为了以下依次执行的阶段：准备、释放、循环、结束。其中，释放阶段与结束阶段是瞬时的，进入该阶段会触发进入该阶段的效果并执行对应的动画模块。准备、循环阶段是持续的，支持被打断并触发打断效果。在循环阶段，还有周期过滤器与生效过滤器，在每个周期中，如果满足生效过滤器，则触发周期效果，如果不满足周期过滤器，则退出循环阶段。

技能在进入准备阶段前会进行条件检查。条件有三：① 单位的体力值、生命值与法力值均满足消耗，且技能的充能次数满足消耗。② 技能的目标满足要求（分别为是否接受自身、是否接受其他游戏对象、是否接受目标点），且目标位于指定范围内。③ 释放者的技能管理器中的剩余准备槽数量满足技能占用的准备槽。

通过对四种阶段的不同执行，技能系统将技能划分为了四个类型：使用型（boss 小雪的多数技能），持续型（暂无实例），开关（如奔跑、走路），被动型（暂无实例）。使用型技能没有循环阶段，释放后直接进入结束阶段；持续型技能释放后进入循环阶段，在循环阶段不满足周期过滤器条件或者持续了指定的时间后，进入结束阶段；开关型技能释放后进入循环阶段，但仅在不满足周期过滤器条件时退出；被动型技能是在技能被添加后就会立刻使用的开关型技能。以上所说的均为自动退出，玩家或者 NPC 也可以手动结束循环阶段，即“停止技能”。

值得注意的时，在实际开发中，为了方便技能释放更加贴合动画，技能的准备更多使用动画事件完成，而非技能资源中的准备阶段。

技能系统的设计经过了许多的更迭才变成了现在的样子，我认为，通过这种阶段的划分，技能系统结合效果系统已经能够设计出至少 95%可以想象的技能了。

主要实现过程：

实现过程如上文所述，基本与设计一致。

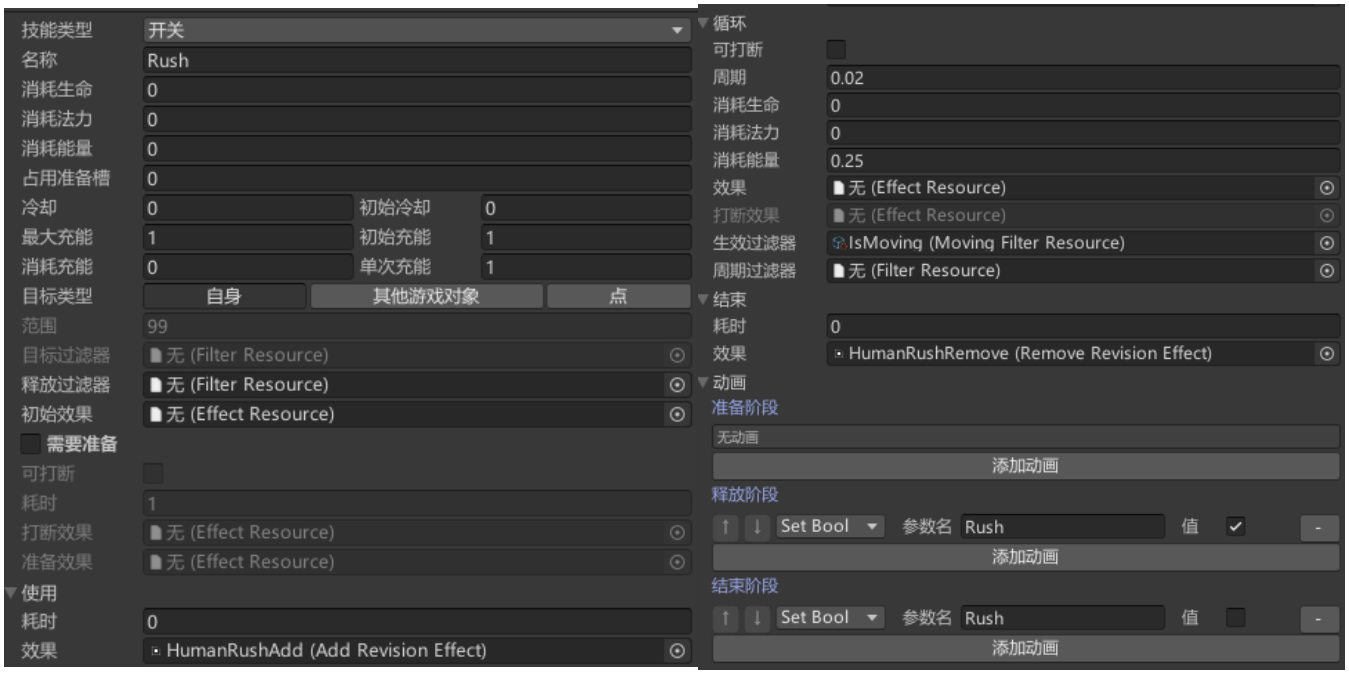

图 8 9 重写的技能资源编辑器界面（图示为冲刺）

八、 武器系统

系统介绍与设计目的：

技能系统是一种自定义资源系统，武器是一种自定义资源，可以替换到单位的武器对象中。值得注意的是，本系统中的武器指的是可以被动态替换的，单位所使用的武器，例如本次主角可以使用的四把武器。而 boss 小雪使用的刀，是直接绘制在人物的图像中的，不使用武器系统。

武器系统分三级管理单位持有或携带的武器，这三级分别是：武器管理器、武器容器、武器对象。

武器管理器负责管理武器系统的激活与禁用，武器的切换，接受指令输入并分发给下级的武器容器；武器容器负责管理武器的旋转与移动，包括闲置时的角度、旋转速度的限制、开火时后坐力的冲击等。武器容器分为两类，携带容器与持有容器，武器对象在携带容器中时，不会接受指令输入，且持有容器中的武器对象在收纳时只会被收纳进入它的初始携带容器。任意收纳容器内仅能收纳它初始拥有的武器对象；武器对象负责武器开火、换弹、过热等的非视觉效果与能量核心颜色、过热颜色的视觉效果，且武器对象支持动态读取切换武器。

武器支持三种扳机类型，分别为压发（半自动）、连续（全自动）与松发（蓄力），压发扳机仅在收到开火指令时开火一次；连续扳机在收到开火指令后会持续开火，直到收到停止开火指令；松发扳机在收到开火指令时开始分等级蓄力，在收到停止开火指令时根据蓄力等级触发对应开火效果。

在目前的游戏开发中，仅使用到了连续扳机。

主要实现过程：

在 unity 中，武器管理器、武器容器、武器对象均作为构件存在，且它们附着的游戏对象也非平级，武器对象的父对象是武器容器的游戏对象，武器容器的父对象是武器管理器。武器在切换时交换两容器的子武器容器游戏对象，同时两容器重新读取武器的角度、距离等属性。

在收到指令后，武器管理器会将指令分发给所有的持有容器，持有容器收到指令后，如果容器内有武器对象，则执行指令并将指令分发给武器对象。

值得注意的是，在收到瞄准指令后切换武器准星是由武器对象负责的，武器对象会检测武器持有者是否拥有玩家控制器，如果拥有，则将武器资源链接的准星资源托管至准星系统（准星系统限于篇幅不做过多介绍）。

剩余未介绍部分实现过程如上文所述，基本与设计一致。

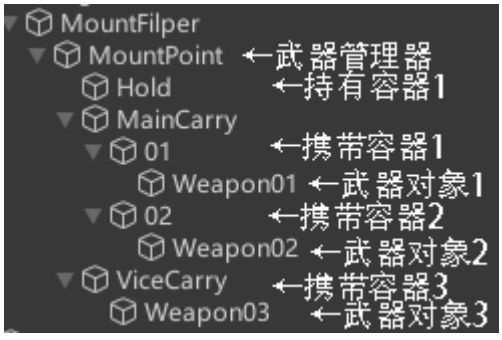

图 10 武器系统上下级示意图

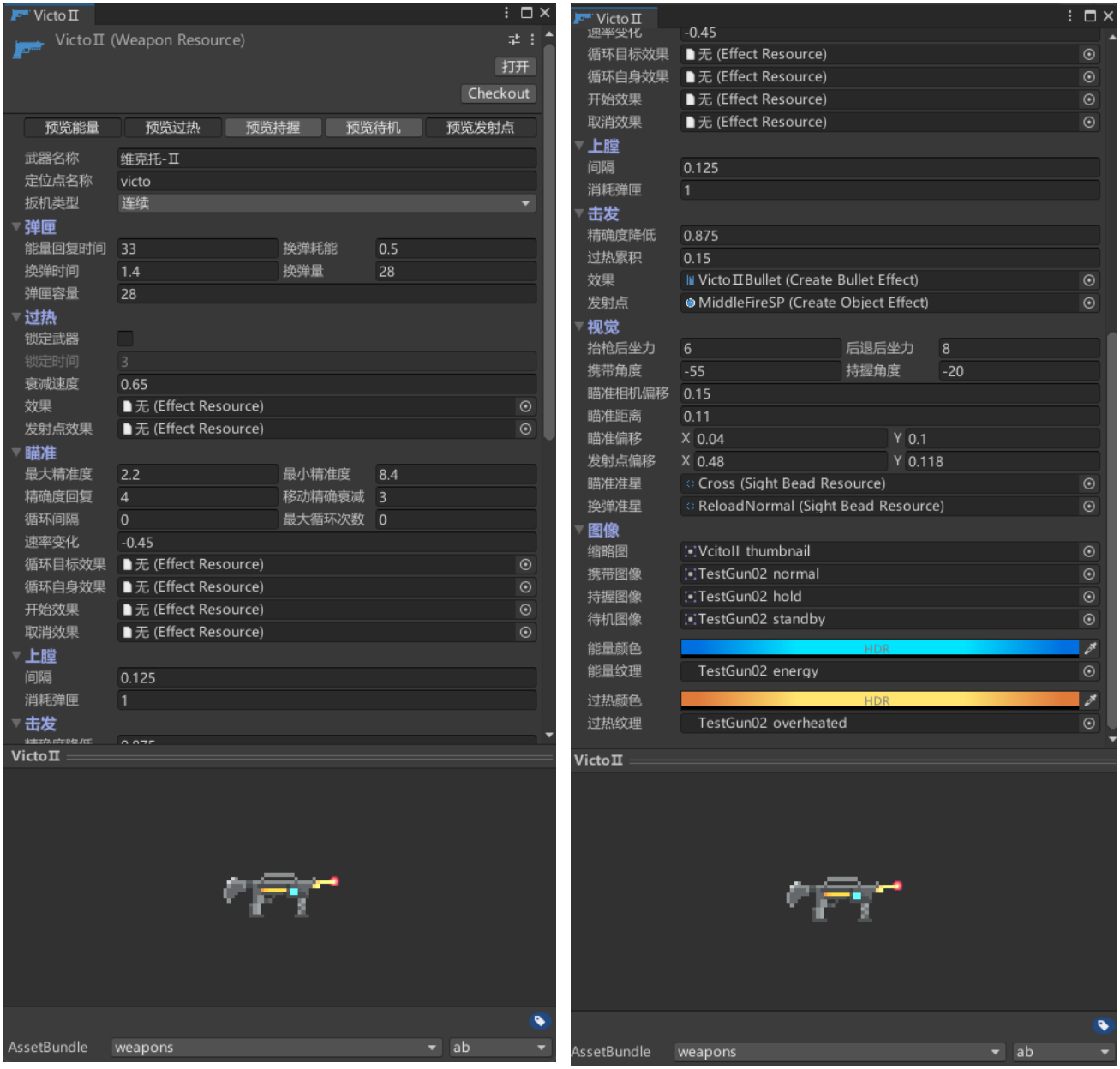

图 11 12 重写的武器资源的编辑器界面（图示为维克托 II）

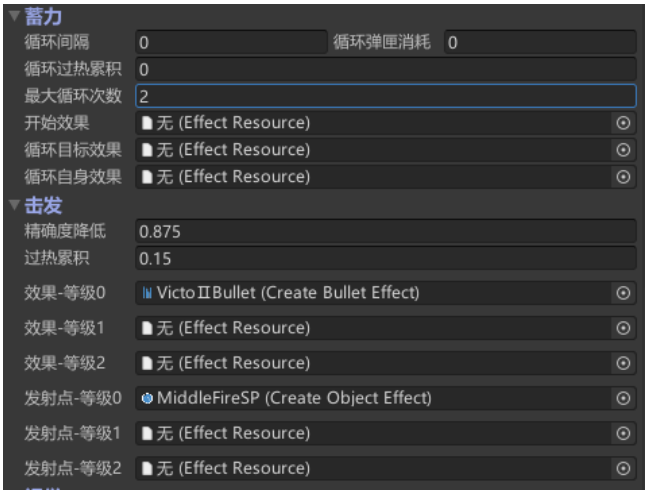

图 13 仅在松发扳机会出现的蓄力阶段与分等级发射效果

九、 通用 AI

系统介绍与设计目的：

通用 AI 是一种构件，附着在 NPC 的游戏对象上。通用 AI 一般作为单位 AI 的父类使用，向子类提供常用 AI 指令的支持。

通用 AI 支持以下方面的指令：搜索目标、危险预知、靠近目标，技能组的定义、索敌与连续释放。当子类启用搜索敌人时，通用 AI 会定期搜索范围内的合法目标并存入敌人列表；当子类启用了危险预知时，通用 AI 会定期检测周围是否存在敌方的子弹，若存在，则触发对应的方法；当子类设置了靠近目标时，通用 AI 会尝试靠近目标至指定距离。

在定义技能组之前，子类需要先定义可使用技能（即会被通用 AI 使用）。可使用技能包含一个单位拥有的技能，释放的目标类型与释放的方向类型。技能组包含若干可使用技能与最小释放距离，最大释放距离，是否可打断等。通用 AI 支持对指定目标随机选择可释放技能组，支持对指定目标释放技能组。

例如，boss 小雪拥有 13 个技能，但只有 9 个可使用技能，这 9 个可使用技能组合成了 12 个技能组。其中，2 个技能组只有一阶段会使用，5 个技能组只有二阶段会使用。

通用AI的设计目的是让单位AI的编写能够将重心放在“这个单位要干什么”，而非“要怎么让他去干”。

主要实现过程：

为了保证效率，通用 AI 的部分方法更新频率仅有 0.2s 一次，包括搜索目标，危险预知与子类 AI 更新。

在设置了靠近目标时，通用 AI 会与单位的通用控制器通信，使用通用控制器向目标移动。

为了满足可能存在的复杂技能索敌逻辑，通用 AI 的技能组还支持传入判断方法来判断目标是否合法，默认情况下，此方法将判断目标是否为敌人。

剩余未介绍部分实现过程如上文所述，基本与设计一致。

```
if (MoveFinish || MoveFrame >= 6)
{
    MoveFinish = true;
    SkillCombo combo = SelectCombo(Target);
    if(combo == null)
    {
        MoveTo(Target);
    }
    else
    {
        UseCombo(Target, combo);
    }
}
else
{
    MoveFrame++;
}
```

图 14 boss 小雪的主要 AI 逻辑

十、 剧情系统

系统介绍与设计目的：

剧情系统是一种自定义资源系统，剧情是一种自定义资源。一个剧情资源主要包含剧情角色、图片与剧情指令。剧情资源可以被剧情播放器播放。

游戏中，所有的剧情均有属于它们的剧情资源，剧情资源还有链接的语言文本，语言文本由语言系统选择读取（语言系统限于篇幅，不做过多介绍）。

在一个剧情资源中，剧情指令是核心，剧情指令会按顺序执行，可以完成包含控制角色，显示对话，控制摄像机，控制时间流速等操作。

主要实现过程：

剧情系统的逻辑设计参考了计算机操作系统中的进程控制。

在一个剧情开始前，剧情播放器会初始化本次剧情中需要用到的资源与它们的资源数量，这些资源包括：单位（例如黑、小雪、琳）、图片、文本、对话框。其中，单位与对话框的资源数量为 1，其余的均为无限。值得注意的是，每一个单位都会有他对应的一个对话框资源，但只有在第一次使用该对话框时，对话框才会被实际创建。

在初始化完成后，剧情播放器会按顺序执行剧情指令。剧情指令分为三个执行阶段，分别为申请资源、执行、结束。剧情指令默认按顺序执行，即在一个指令结束后再执行下一指令，但剧情指令也支持并发执行，即一个指令进入执行阶段后马上开始下一指令的申请资源阶段。

剧情指令的申请资源是一个原子操作，如果申请资源不足，则等待任意执行中指令执行完毕释放资源后再尝试申请资源。如果某指令在没有其他执行中指令后申请资源仍然失败，则该指令会被标记为非法指令并忽略。

目前支持的剧情指令如下：

```
/* Command List
 * timescale <timescale> <time>   缩放时间
 * cameraspeed <rate>    调整摄像头速度
 * camerafocus <character> <offsetx> <offsety>    摄像头聚焦
 * camerasize <size>    调整摄像头视野
 * camerashake <shakerate> <time> <speed>    摄像头抖动
 * wait <time>    等待指定时间
 * move <character> <character> <offsetx> <offsety> <range>移动单1位至单位2的相对坐标
 * move <character> <positionx> <positiony> <range>        移动单位至坐标
 * face <character> <character>    单位1转向至单位2
 * face <character> <l/r>    单位转向至指定方向
 * stare <character> <character>    单位1持续注视单位2
 * nostare <character>    单位停止注视
 * useskill <character> <skill>    单位对自己使用技能
 * useskill <character> <skill> <character>    单位1对单位2使用技能
 * stopskill <character> <skill>    单位停止技能
 * autodialogside <character> <character>    自动调整两单位的对话框的推荐方向
 * setdialog <character> <sprite> <textkey> <l/r>    设置某单位的对话框
 * setdialog <character> <sprite> <textkey> <-r->  设置某单位的对话框，推荐方向默认向右
 * dialogside <character> <l/r>    设置某单位的对话框推荐方向
 * dialogtext <character> <textkey>    设置某单位对话框的文本
```

```
 * dialogimg <character> <sprite>     设置某单位的对话框的图片
 * hidedialog <character>     隐藏某单位的对话框
 * showdialog <character>     显示某单位的对话框
*/
```

图 14 重写的剧情资源编辑器（图示为进入大门后剧情-旧）

十一、状态系统

系统介绍与设计目的：

状态系统是一种自定义资源系统，此系统的目的是实现现在游戏冲常见的增益、减益或者中性的状态。

状态本身作为一个对象被附着单位上的状态管理器集中管理，集中更新与集中移除。

状态的属性分为基础属性与层数属性，基础属性控制状态层数的自然更新（如随时间递减），层数属性会根据层数的不同表现出不同的效果。层数的变化还可以触发一些额外的效果。

此系统设计目的是能让游戏技能、攻击机制的设计多样化，不受限于简单的数值加减。

主要实现过程：

状态的更新由单位身上的状态管理器集中管理，状态自身拥有层数，且可以在不同的层数反应出不同的效果。

状态可以添加事件扳机到游戏的单位事件管理器中，在事件触发时来完成存储在状态中的事件过程，在此过程中可以阻止诸如受到伤害、造成伤害、死亡、使用技能的事件的发生，或者在同时实现一些其他的效果（例如，游戏技能盾牌反击就添加了一个判断方向的受到伤害事件扳机）。

状态可以拥有一个贴附在单位身上的游戏对象来实现状态的特效，并通过事件来进行动画的控制。

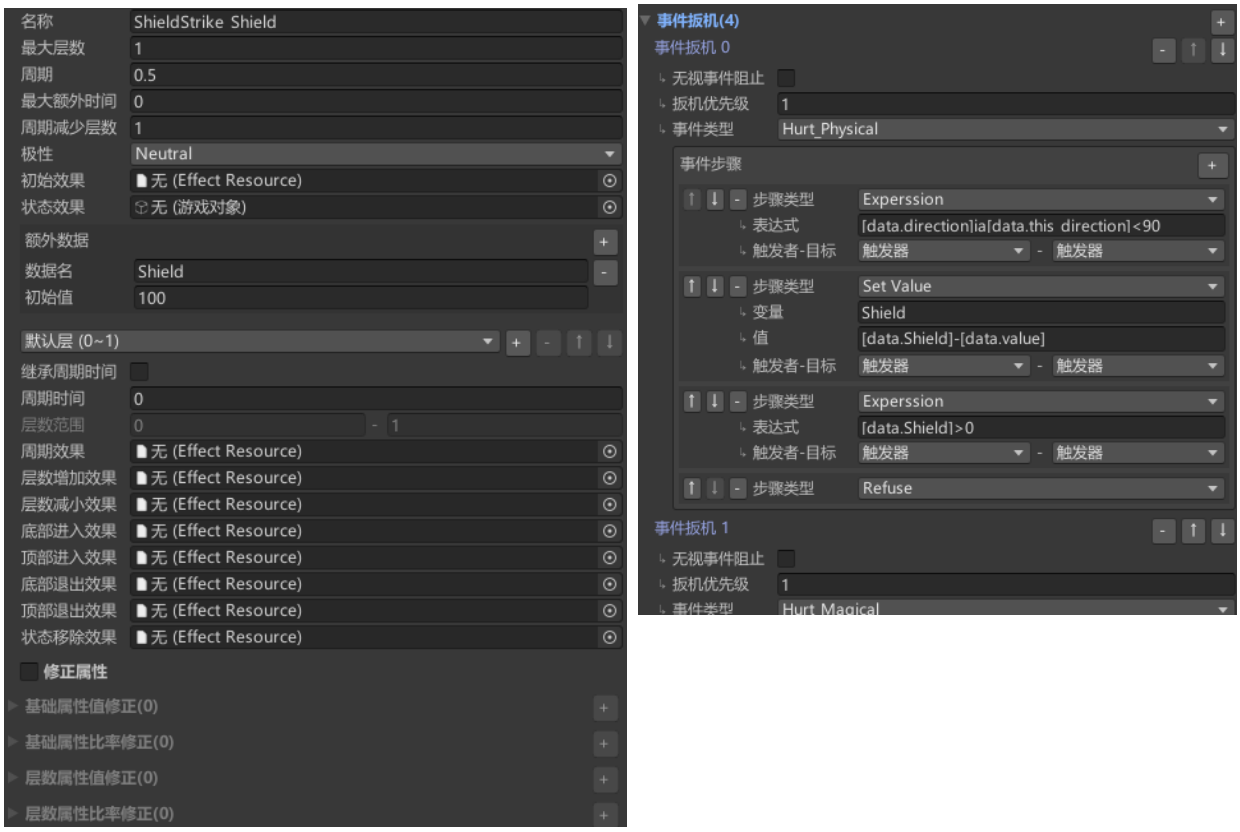

图 15-16 重写的状态编辑器（图示为盾牌反击技能的部分状态内容）