YY33技能系统设计

技能系统整体设计

设计目标

简而言之，我希望技能系统能够达到以下三个目标：

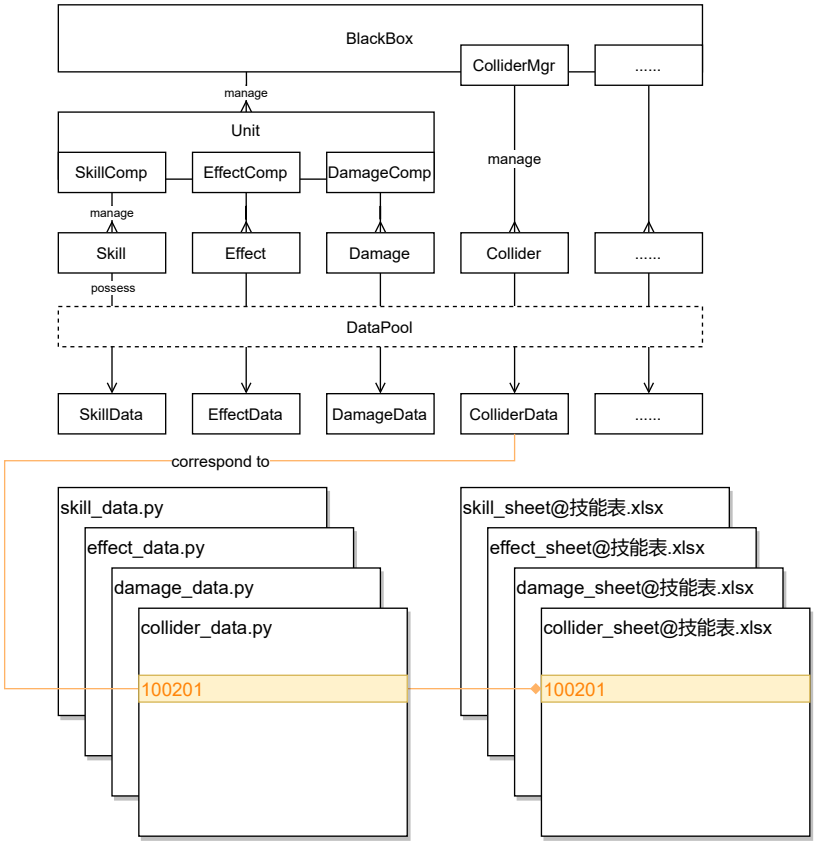
- 轻量级，易扩展
- 模块可复用，减少冗余代码
- 代码风格统一，逻辑一致

对象分层

从对象和数据的层次上来说，技能模块整体分为三层：

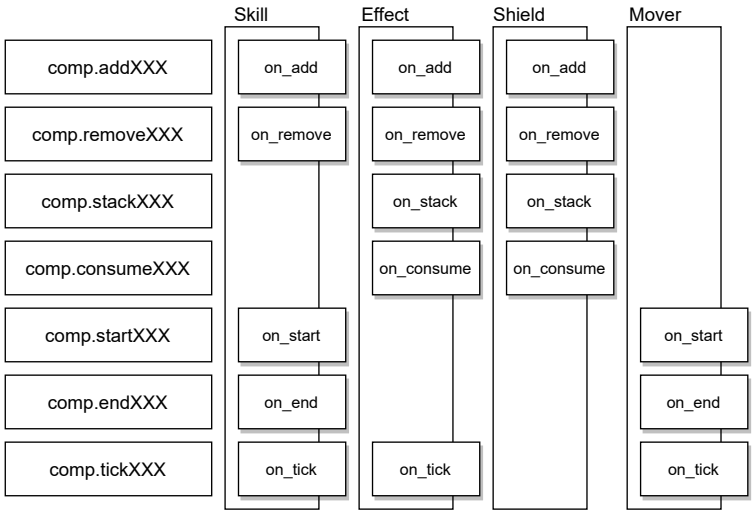
- 静态数据层。对应于导表文件中的一行，其源于策划在excel或者技能编辑器中的配置项
- 运行时对象。其通过数据池持有一份静态数据，本身持有动态数据，动态的数据会在运行时变化
- 管理类。通常是`Unit`或者`BlackBox`（相当于逻辑上的`GameWorld`）上的`Component`，用来管理对应的运行时对象

部分对象的层次图如下图所示。

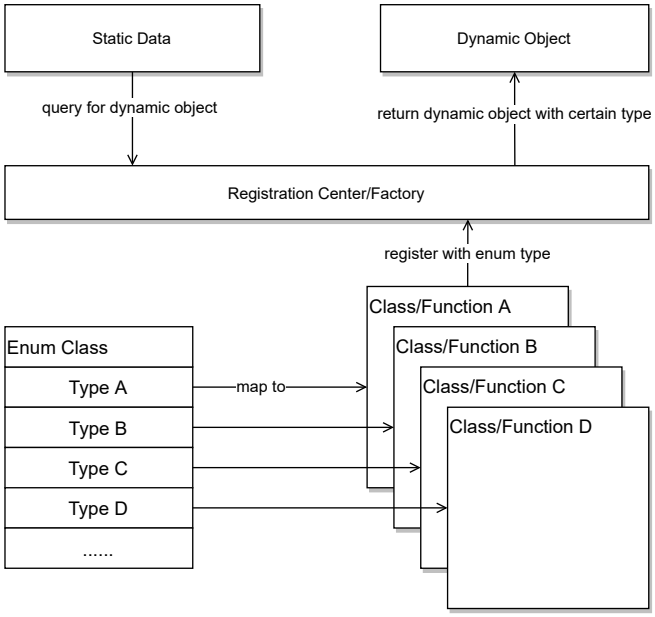


统一的模式和接口

在实现不同的技能模块的时候，我观察到了一种通用的模式，实际上也就是对象的生命周期，这与安卓开发中`activity`的生命周期的概念相似。对象会产生，消亡，开始，结束，增加，消耗，几乎所有的技能对象都能够使用类似的方法来管理，这使得模块间的代码风格统一。



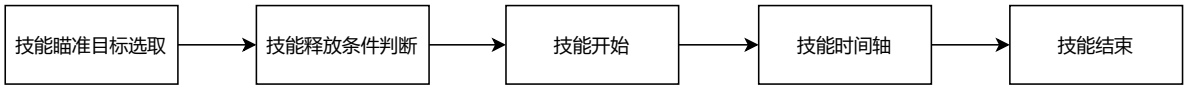
此外，结合导表定义的工厂模式也在技能系统中广泛使用。这通常使用在有多种子类的对象中，例如各种运动的轨迹，不同形状的攻击盒，以及不同类型的触发效果等。对于这一类对象，在静态数据中通常会有一个字段表示其类别，其导表定义对应于程序中的`enum`类，从而能够利用工厂类来生成运行时对象；在程序一侧，给这些类加上统一的装饰器能够方便地将多种子类注册到工厂类中。



技能系统的各组成部分及设计

主动技能

主动技能通常是玩家/AI主动使用，首先技能会选择一个瞄准目标，在满足一系列释放条件后技能开始，并进入技能的时间轴，在技能时长结束或者被外部因素打断时中止。



瞄准目标选取

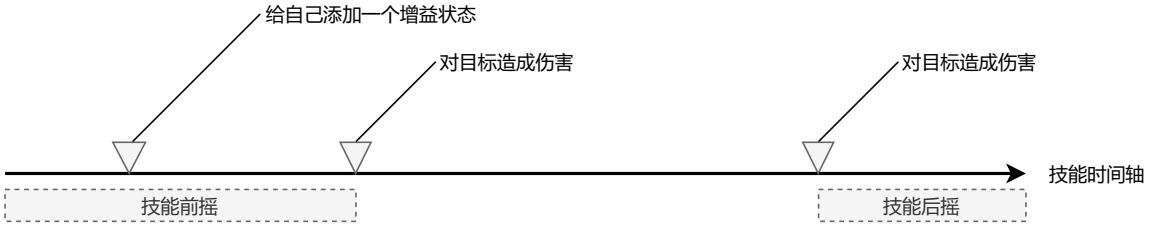
瞄准目标的选取在游戏中是AI来完成的，目前有三种瞄准目标类型：

- 选择一个战场单位
- 选择一个位置
- 选择一个方向

选择后的瞄准目标会存在技能对象中，供技能时间轴上的触发事件使用。

技能时间轴

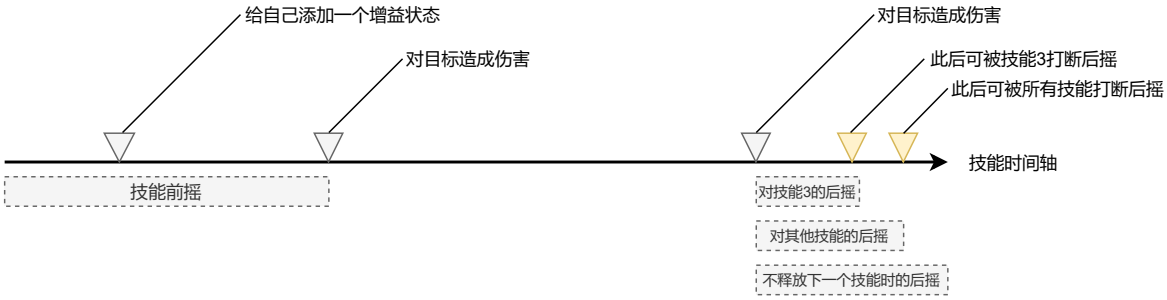
时间轴是主动技能所产生的技能效果的抽象。时间轴的长度与该技能的动作相同，通常策划也是对着技能动作逐帧配置伤害帧等技能效果的时间。时间轴上可配置的触发效果是一系列封装好的，可自由组合的条件和效果函数。



技能结束/打断

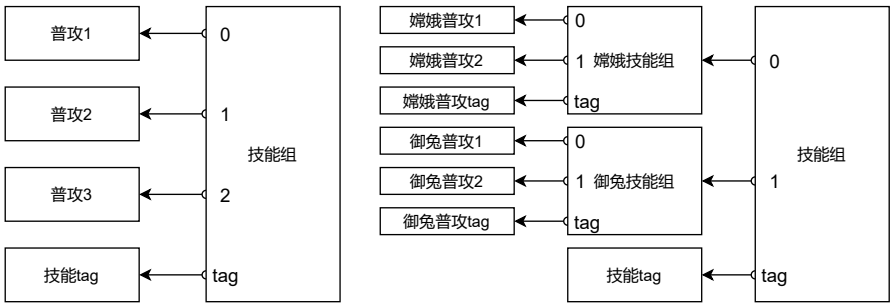
技能的打断和结束都是动作和时间轴的停止和复位，因此实现上基本一致，时间轴在停止后后续的事件不再触发，从而达到逻辑和表现的一致性。通常来说，技能被打断都是源于外部因素，例如其他单位的控制性技能；技能结束主要都是来自自身因素，包括达到技能时长的正常结束，以及施放超必杀等高优先级技能时也会结束上一个技能。

此外，策划考虑到技能衔接的流畅度，会希望在不同技能前后释放时，前一个技能的动作提早结束，立即放出下一个技能，也就是打断技能后摇。由此引出了一个额外的技能结束机制，程序中命名为`supercancel`点。`supercancel`点可以针对特定技能配置，通过安插多个`supercancel`点，能够实现对不同技能配置不同的后摇。



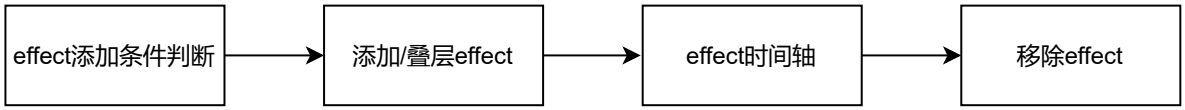
技能组的概念

常见的`moba`类游戏中，通常都会有多段技能的概念，即同一个按键点击多次会释放不同的技能，最典型的例子就是英雄的普攻，其会在多个动作间顺序切换。为了实现这一类需求，我引入了技能组的概念，技能组是一类虚拟的技能，相当于技能的选择器（`selector`），其根据当前的技能tag选择其中的一个子技能来释放。同时，技能组的实现也支持嵌套，从而能够配置英雄在两套技能组之间切换，例如YY33中能够变身的英雄嫦娥就使用了嵌套的普攻技能组。



被动技能与BUFF

在H52的技能系统中，被动技能与BUFF是两个独立的模块，但是在开发的过程中会发现，有大量代码是可以复用的，甚至被动技能本身也可以看作是一个常驻的BUFF。因此，在YY33的技能系统中，被动技能与BUFF的实现方式是相同的，程序中将其统称为`effect`。



`effect`的第一个主要作用是能够方便策划进行时间相关的操作，例如想要将某个单位眩晕5秒，或者是增强某个单位的攻击力持续3秒，策划通常都是通过配置一个对应时间的`effect`来实现；effect的第二个主要作用是完成一些特定时机触发的效果，例如想要在受到伤害时对敌方反伤，或者是使用技能后强化下一次普攻，这些都能够通过`effect`来配置。

effect的添加与叠加

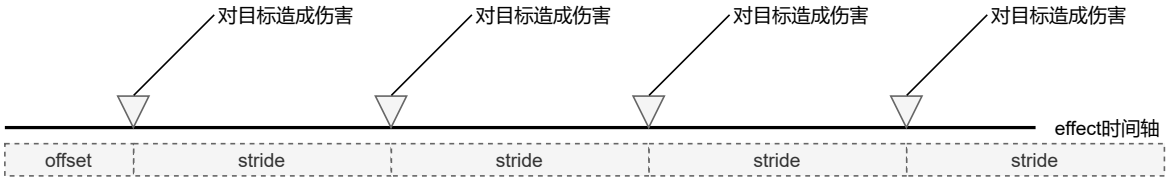
当单位A试图向单位B添加一个编号为C的effect时，如果B身上此时还没有编号为C的effect，那么就触发了effect的添加（add）；如果此时B身上已经有了编号为C的effect，那么就触发了effect的叠加（stack）。YY33的技能系统中定义了以下几种叠加规则：

- 覆盖。使用新effect的等级
- 叠层。使用两者等级之和
- 合并。使用两者中等级较高的

目前时间上的叠加统一采取了两者剩余时长中取长，根据需求可补充新的时间叠加规则。

effect时间轴

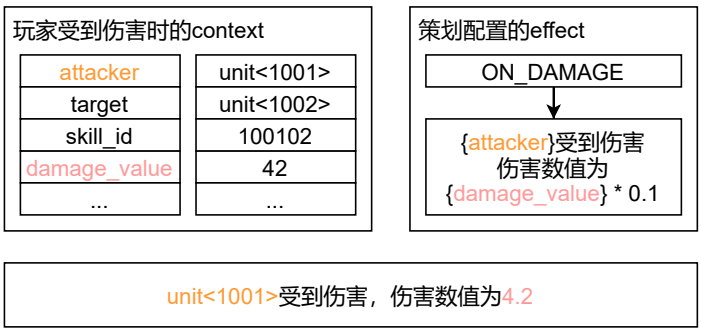
相比与技能的时间轴，effect的时间轴并不对应于某个动作，因此即使对于同一个effect来说，其时间轴的长度是任意可变的；除此之外，effect的时间轴是可循环的（例如每0.5秒造成灼伤效果的debuff）。根据经验，effect大多都使用的是循环的触发效果，但同时也有一定数量的一次性的触发效果，在程序中通过一个第一次触发的时间（offset）和循环间隔（stride）加上循环次数（count）即可描述一系列循环的时间轴上的点。



effect中的被动事件与context

effect中的被动事件通过观察者模式实现，配合生命周期中合适的埋点触发。例如，在玩家受到伤害时，会发出一个玩家受到伤害的事件（`send_event(ON_DAMAGE)`），对应的effect能够接收到该事件并触发策划所配置的技能效果。通常在某个事件产生的时候，我们需要当时的上下文信息，例如攻击者，攻击者使用的技能，伤害值等等，这些信息会在事件发出的同时被打包成一个context传递到接收者（通常是effect）中。

context类实际上就是一个dict的派生类，其中简单地封装了一些公式解析相关的方法，因为context中的数值大多都是在策划填写的公式中所使用的。一个简化了的产生反伤效果的被动事件配置和流程大致如下。

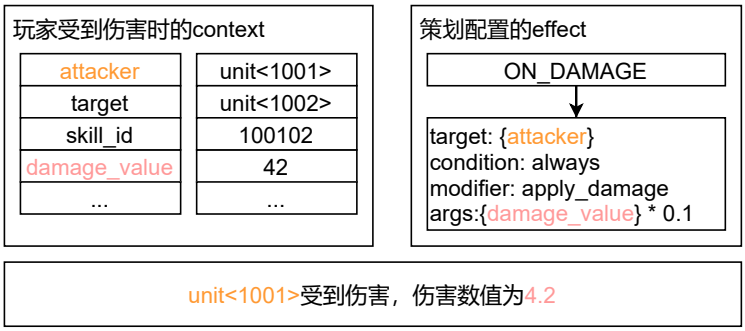


触发效果

程序中的触发效果是策划可调用的方法的集合，前文中所使用的**触发效果**一次均指代该模块中的内容。这一模块统一了技能时间轴，effect时间轴，被动触发事件等等所有能够触发的技能效果，实现了技能效果的复用，这也是这个模块设计的核心目的。一个触发效果通常包含以下部分信息：

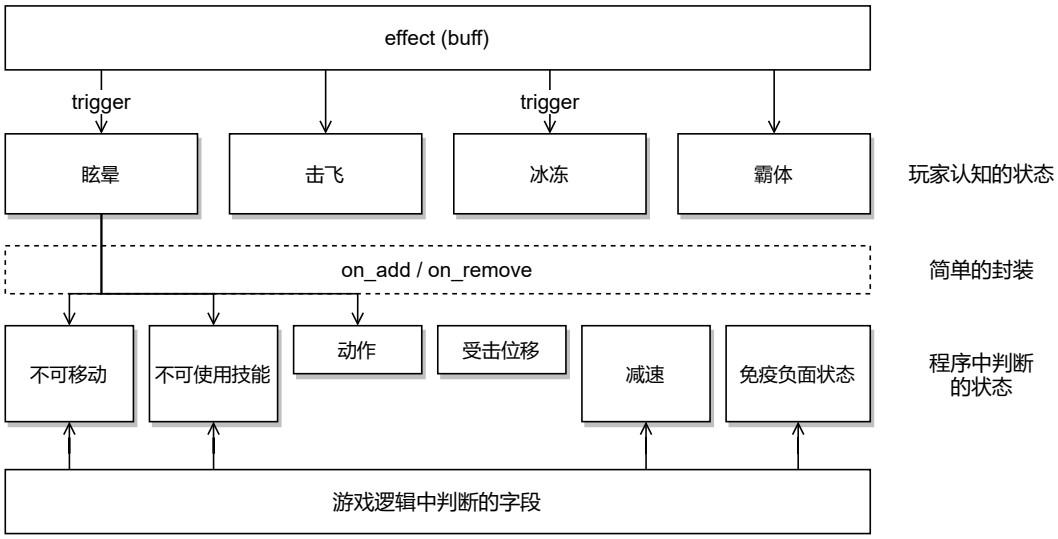
- 谁来触发，即触发者`target`。常见的触发者包括攻击方/受击方等等。但是触发者并不一定局限于战场中的单位，其理论上能够是任意的对象
- 在什么条件下触发，即触发条件`condition`
- 触发什么技能效果，即真正的触发函数`modifier`以及附带的参数`args`
- 触发时刻的上下文`context`，由上层传入

前文中简化了的反伤效果的被动事件，其更接近真实的配置方式如下图所示，其中的`args`所代表的公式会通过前文中所述的`context`中的变量计算出真正的值。



单位状态管理

单位的状态通常都是由effect附加的，我在最初的实现中并没有增加玩家认知的状态层，而是在`effect`的`on_add`和`on_remove`中直接调用触发效果或者更改标志位，但是这样就会丢失玩家所认知的状态信息，同时，程序中判断单位的状态变得不够直观。因而最后多加了一个简单的状态管理组件，因为我倾向于程序中的实现逻辑与技能的描述尽可能一致，在逻辑自洽的情况下出现的问题（甚至不能称之为问题），至少是玩家能够理解的。



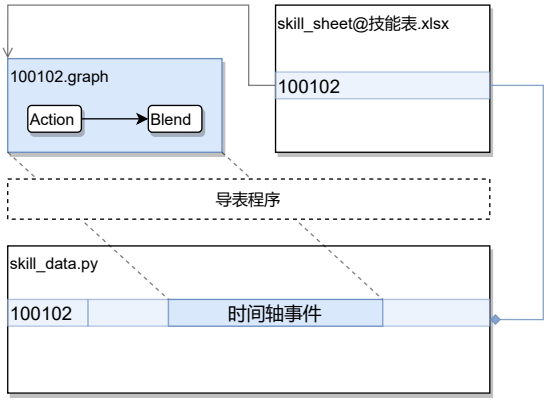
技能编辑方式

excel + graph时间轴

目前的技能主要还是通过excel来配置，其优点在于很容易在多行之间进行比较，也就能够同时编辑修改多个技能。但是excel的问题在于，随着配置项目变多，excel的表头，也就是列数量会不断增长。我们项目中尝试利用excel中的一些工具来控制列数量，例如通过下拉框明确填写范围，通过自动生成的填写提示简化填写成本等。尽管如此，使用excel来配置技能还是较为复杂。

E	F	G	H
谁来触发？	触发函数	触发函数参数	备注[括号内为可选填写]
unit_selector	modifier_id	modifier_args	#ps
int{无:-1,触发器所有者:0,目	int{空函数:0,创建攻击盒:1,受	list<f_value_fix32>	str
锁定目标	受到伤害	ID(100101_1)	ID(伤害编号)
触发器所有者	切换技能组tag	100101,1,192	技能编号,切换到的tag,[TM(复位时间)]
触发器所有者	切换技能组tag	100101,0,60	技能编号,切换到的tag,[TM(复位时间)]
触发器所有者	添加buff	100151,1,-1	buff编号,buff等级,TM(buff时长),[buff来源方]
触发器所有者	子弹时间	100151,5	TM(时长),[其他单位减速倍率]
触发器所有者	消耗buff	100151,1	buff编号,buff等级,[buff来源方]
触发器所有者	受到治疗	ID(100151_1)	ID(伤害编号)
触发器所有者	创建攻击盒	(100102_1)	ID(攻击盒编),[攻击盒目标信息]
受击方	创建攻击盒	(100102_1)	ID(伤害编号)
触发器所有者	受到伤害	(100103_1)	ID(攻击盒编),[攻击盒目标信息]
触发器所有者	切换技能组tag	(100103_2)	ID(攻击盒编),[攻击盒目标信息]
受击方	添加buff	(100103_1)	ID(伤害编号)
受击方	移除buff	(100103_2)	ID(伤害编号)
触发器所有者	触发位移	(100104_1)	ID(攻击盒编),[攻击盒目标信息]
受击方	修改属性	(100104_1)	ID(伤害编号)
触发器所有者	触发其他trigger	100000,1,TM(1.77)	buff编号,buff等级,TM(buff时长),[buff来源方]
触发器所有者	添加buff	100000,1,TM(3.43)	buff编号,buff等级,TM(buff时长),[buff来源方]
触发器所有者	创建攻击盒	ID(100105_1)	ID(攻击盒编),[攻击盒目标信息]
触发器所有者	创建攻击盒	ID(100105_2)	ID(攻击盒编),[攻击盒目标信息]

对于策划来说，通过excel配置技能最难以接受的是时间轴事件，因为策划配伤害帧通常都是要配合动作来看，这需要反复播放动作进行调试，因而提出了将触发效果作为cue事件放在graph上的想法。然而，在YY33的帧同步框架下，是不允许表现来驱动逻辑的，因此只是利用了graph作为数据的配置方式，而是通过导表程序，将技能对应的graph中的cue数据导出到py数据文件中的时间轴事件一列。



pyflow编辑器

pyflow技能编辑器由H52的工具组其基于开源的pyflow框架开发，其类似Unreal中的蓝图，能够通过节点图的形式自定义程序的执行流，同时其能够根据节点图生成对应的代码。然而在YY33项目中，个人认为不适合直接通过生成的代码来管理运行时的对象状态。因而，在尝试接入pyflow编辑器的时候，目前是通过程序中的各个埋点来调用pyflow编辑器生成的函数。
pyflow的问题在于目前的对策划来说的编辑难度较高，因为执行流的节点图带有比较强的程序思维，对于策划来说反而是负担；此外，pyflow的ui框架原本作为一个用于模拟程序执行流的ui框架，对于时间轴/子图等游戏常用的配置方式不够友好，需要自行来扩展相对应的功能。

polaris编辑器

polaris技能编辑器是sunshine编辑器中的一个插件，其在编辑方式上专为技能编辑而适配，相对pyflow编辑器较为友好一些，目前正在预研阶段尝试接入。

小结与未来的改进方向

技能编辑器
更纯粹的ECS