**总览**

核心：

1行为和数据

2 Actor = Player + Npc 的基类

什么是玩家和NPC共有的属性和方法

EActorType = {Player, Npc}

中间类负责各种Actor之间的交互

如果是Actor之间则传入参数为(Actor, Actor)

如果是Player和Actor之间，则传入参数为(Player, Actor)

Actor基本属性

Lv

Str

Dex

Int

Spi

Cun

Vit

BasePhyMeleeMax

BasePhyMeleeMin

BasePhyMelee

PhyMeleeMax

PhyMeleeMin

PhyMelee

BasePhyRangeMax

BasePhyRangeMin

BasePhyRange

PhyRangeMax

PhyRangeMin

PhyRange

BaseMgiMax

BaseMgiMin

BaseMgi

MgiMax

MgiMin

Mgi

FireDamageMax

FireDamageMin

FireDamage

IceDamageMax

IceDamageMin

IceDamage

LightningDamageMax

LightningDamageMin

LightningDamage

PoisonDamageMax

PoisonDamageMin

PoisonDamage

ArcaneDamageMax

ArcaneDamageMin

ArcaneDamage

ShadowDamageMax

ShadowDamageMin

ShadowDamage

HolyDamageMax

HolyDamageMin

HolyDamage

Ias

CritCha

HPMax

**Data脚本**

Data（数据）：保存处理所有的属性，数据，战斗，Buff/Debuff

所有属性和字段

DealDmg()

TakeDmg()

AddHP() \*判断生命是否溢出

SubHP() \*之后需要进CheckDead()

CheckDead() \*判断是否死亡

主角：

造成伤害的方法DealDmg()

接受伤害的方法TakeDmg()

NPC：

造成伤害的方法DealDmg()

接收伤害的方法TakeDmg()

主角释放技能or攻击时，通过各种手段获取受击目标的List

遍历List中所有GameObject，调用这些GameObject的Data脚本

1造成伤害方法DealDmg()

2受到伤害方法：

TakeDmg（int dmg, enum dmgType,bool isDirectDmg,GameObject dmgSource,Struct PeneValue）

2.1传入参数：

-收到的伤害数值

-收到的伤害类型Enum 物理/魔法/各种元素/纯粹（真实伤害）

-是否是直接伤害（直接伤害可以被格挡/精准闪避，打断持续性恢复效果等）

-攻击来源GameObject（用于判断反击）

-各种穿透属性的结构体 Struct PeneDataStruct，其中PeneValue[0]固定护甲穿透，PeneValue[1]固定抗性穿透，PeneValue[2]百分比护甲穿透，PeneValue[3]百分比抗性穿透

-（扩展…）

2.2脚本逻辑

-判断是否无敌，如果无敌直接返回false，Evt OnPlayerHit(0，eInvulnerable)

-判断是否闪避，如果闪避直接返回false，Evt OnPlayerHit (0，eEvade)

-Warrior独有- 伤害来源朝向判断，如果来自正面120°/180°则计算伤害减免

-伤害数值计算，基于伤害类型Enum和自身护甲/魔抗

-判断是否格挡，如果格挡计算伤害减免Evt OnPlayerHit (0，eBlock)

如果完全格挡伤害返回false，否则返回true

-Mage独有- 是否处于法力护盾，如果是先扣除护盾承受伤害，计算剩余伤害

-是否有护盾，护盾承受伤害之后剩余伤害，护盾是否在这次之后被击破Evt OnPlayerHit (0，eNormal)

返回true

-Warrior独有- 是否处于防御状态，如果处于防御状态作出处理

（先获得伤害减免计算，然后扣除防御值）

防御值是否被击破，Evt OnPlayerHit (0，eNormal)

返回true

承受伤害，调用方法SubtractHP (int realNumber)

2.3事件和回调

OnPlayerHit()

-作用是如果有监听受伤事件的脚本，需要在此时做对应的处理

int takenDamage 最终承受的伤害

OnPlayerMagicShieldTakenDmg(int Dmg)

-Mage魔法盾承受伤害

OnPlayerMagicShieldBroken()

-Mage魔法盾被击破

OnPlayerShieldBroken()

-护盾被击破

OnPlayerGuardBroken()

-Warrior防御值被击破

2.4特殊处理

**Behaviour脚本**

核心行为逻辑

Behaviour（表现）：保存处理所有的输入指令、行为树、动画状态机

Step1 InputCheck玩家输入

如果可以输入 更新玩家的输入

H = 水平方向轴值

V = 垂直方向轴值

玩家Input类单独隔离

\*\*\*设计原则\*\*\*

同一时间只能进行一个Action

改变相应的状态

和Action相关的

bInAction

bCanBreak

如果bInAction，检查bCanBreak

Step2 StateCheck是否在各种状态

在这里仅做状态的

ActionStart

ActionOver

读取检查，不做任何修改行为

bCanBreak可以被打断

bIsStill 完全静止，取消重力

bIsDead 是否死亡，

Layer2/3/4权重为0，SetTrigger（Die）

(在玩家的Data脚本中拥有死亡回调，死亡回调会设置这些状态，bIsDead = True, bCanControl = false)

bIsCharging 正在蓄力

bIsSinging 正在吟唱

bIsChannelling 正在通道

bLockRotate 锁定方向，如果bLockRotate则无法进行旋转

bForceCamForward 强制面对镜头前方，用于制作冲锋

bLockGravity 锁定重力

bNoGravity 不受重力

int Gravity 设置重力，配合LockGravity使用

bLockSpeed 锁定速度

int MoveSpeed 设置速度，配合LockSpeed使用

Step3PlayerInput玩家输入

Rotate()

Gravity()

Jump()

Move()

RotateToCameraForward()

Action.CheckAction()在这里检查是否释放技能

需要Frame刷新的事

重力

旋转

移动（必须先旋转再移动）

校准（必须在移动旋转之后）

行动Action

**NPC Behaviour类**

Step1是否在战斗状态bCombatStatus

\*玩家逻辑

战斗状态如果不在任何怪物的CombatList中则为非战斗状态，否则为进入战斗状态

补充，战斗状态会影响Data脚本处理，比如是否可以潜行等

\*怪物逻辑

不在战斗状态，执行非战斗状态脚本

如果在战斗状态，执行战斗状态脚本，首先获取仇恨目标

Step2是否可以做出行动，检查通用型Debuff(受控制效果)

CheckBeenControlled()

\*玩家逻辑

如果有行动能力，则Input有效，否则Input无效

如果进入了释放技能中控制受限状态，则做出对应处理

\*怪物逻辑

如果有行动能力，执行行动脚本，如果没有行动能力，则直接结束

Step3怪物行动行为

设置行动相关参数：移动速度等

执行行动脚本，获取进攻模式

通过NavMeshAgent实现进攻模式

???

**技能和动作ActionBase**

几种类型的技能

## 调用顺序：

CharacterBehaviour -> CharacterAction -> ActionBase -> ActionManager -> AnimationEvent

**CharacterBehaviour(Update Start)**

来自一连串的每帧Update判断，在其中的一个步骤监听按下了什么按键，是否可以释放技能

每帧调用CharacterAction对Action进行更新

**CharacterAction**

更新Action状态，如果CharacterBehaviour有Action输入，（TODO）则进行初判断，能否释放技能

更新Animator的权重等

**ActionBase**

拥有不同时机的函数，播放动画，触发ActionManager的技能事件进行广播

1广播给其他关系的脚本，因为广播来源仅此一处，其他脚本只需监听此一处

2设置了Animator的Trigger，开始播放动画

**ActionManager**

拥有不同时机的事件，所有Action都通过它进行事件广播

所有关注事件的脚本，监听它的事件即可

**AnimationEvent**

由动画片段触发，动画片段支持四个参数

Float 不填

Int 填Action的ID

String 不填

Object 绑定玩家本身

一旦OnActionEnd(int ID)触发，会直接调用ActionBase的脚本

释放完成CastEnd，技能生效CastLaunch都是由ActionBase的事件首先发起的

\*热键 不同于按键 在InputManager中可以设置热键绑定具体键盘按键

Update里刷新，绑定热键OnKeyPressed()

多重按键优先级，Tab>Shift>Crtl>Q>E>R>1>2>3>4>Right>Left

Step1预检查，检查魔法，检查CD，如果满足进入下一步，任一不满足return，调用UI

--按下按键

ActionPreCheck(action)

EvtActionPreCheck(obj sender, EventArgs info)

--开始吟唱

SingStart(action)

--结束吟唱

SingOver(action)

--开始Anim

ActionStart(action, numCount)

--Anim生效点（可以有多个,参数为第几个生效点）

ActionCast(action)

--结束Anim

ActionOver(action)

enum ActionPreCheckType

{

success,

cdFail,

costFail,

slienceFail,

otherFail,

}

统一为public enum OnActionPreCheck()

在UI-Hint-Warning上注册事件

OnTryStartSkillFailedCD() 做出对应UI效果

OnTryStartSkillFailedEnergy()做出对应UI效果

Step2预检查通过之后，释放前时机，OnBeforeSkill()释放前时机

开始进行Animator相关，SetTrigger，SetLayerWeight

player.data.ChangeEnergy()改变能量

Step2开始蓄力，OnChargeStart()

Step3结束蓄力，OnChargeOver(nChargeTime, bFullyCharged)

Step4开始释放，OnSkillStart()

setCD()改变CD

bIsSkill = true

创建一个Process

不同类型

1仅影响自我

2自身中心AOE/自身中心扇形

3召唤Dummy

Step5释放结束，OnSkillOver()

改变动画层权重（差值计算）

**Channel和Cast施法的区别**

**任何一个CastLaunch和ChannelLaunch都算所技能生效**

Cast直接施法，就是普通施法，播放一段动画

Channel通道施法，其实和施法很像，只有一些表现上的区分

UI表现，被打断之后的处理

OnActionCastStart

OnActionCastLaunch

OnActionCastEnd

OnActionChannelStart

OnActionChannelUpdate

OnActionChannelLaunch

OnActionChannelEnd

释放分类Enum

1直接释放 Cast

2蓄力释放Charge

3吟唱释放Sing

技能模板类型

1仅影响自我

2自身中心AOE/自身中心扇形

3召唤Dummy

1造成伤害

DealDmg()

2造成效果

DealEffect()

3创造虚拟体

CreateDummy()

4自身处理

SelfEffect()

ActionManager 类

每一个按键对应一个ActionBase基类

DoAction()

enum CurrentAction 当前正在进行的动作

BreakAllAction() 打断所有动作，如果非免疫打断，会被执行ActionOver

ForceStopAllAction() 强制停止所有动作，并强制执行当前动作的ActionOver

ActionBase 基类

方法

这里的方法区分了不同的技能释放时机

1.1ActionPreCheck() 检查阶段

1.2返回检查是否通过，并调用下一步操作（2.1）

2.1ActionStart() 动作开始 播放动画

2.2OnActionStart()扣除资源，设置CD

3.1ActionCast() 动作完成 通常在这里设置 bInAction = false

3.2

4.1ActionOver(bool Break) 动作结束

4.2

是否是因为被打断结束的 这个方法可以被别的地方调用

需要一个通用方法，终止一切动作

ActionSelfAOE

+Range

+Angle

+GameObject SphereDetect(){ }

+void ClearDetectGroup(){ }

自身AOE技能模版，例如战争践踏

CastType：Default

PreCheck正常流程

PreCheck通过之后，自动进ActionCastStart()

--

常用方法,可以从其他Manager脚本调用·

EffectManager.AddEffect 增加效果

CharacterBehaviour 设置各种属性

CharacterEffect 设置各种Effect效果

Character

属性（依赖于Excel读表）

int ID 动作ID

string Name 动作名

string Des 动作描述

string Img 动作图标名(自动拼路径 Asset/Image/SkillIcon)

float CoolDown 冷却时间

float EnergyCost 能量消耗

float EffectRatio 触发系数

bool canNotBeBreak不可被打断（优先级最高）

bool canBreakOther 可以打断别的动作（除了“不可被打断”的动作无法打断）

bool canCastWhenControlled 是否可以在被控制时使用

enum eSkillCastType

eSkillCastType.Immediate 瞬间释放,没有任何动作

eSkillCastType.Normal 普通

eSkillCastType.Charge 蓄力

eSkillCastType.Sing 吟唱

运行时属性（运行时产生）

bool bInProgress 这个动作是否正在进行中

float ActionTimeStamp 释放动作的时间戳

**动画树**

NPC：

攻击行为分解：

1准备

2出击（精准闪避）

3判定Frame（真正造成伤害）

4硬直

\*如果敌人正处于出击阶段，且玩家在攻击范围之内使用了翻滚，被认为是进行了一次精准闪避

不同层级

层级最小权重0.001，为0会导致代码失效

1 Base层

2Action层

Default中间状态：Default

类型1 DirectAction 直接

类型2 ChargeAction 蓄力

类型3 SingAction 吟唱

类型4 None 无动作（这种情况没有必要写一个动画状态）

**通用BUFF/DEBUFF**

EffectManager SingletonClass

Buff/Debuff 都被认为是Effect

fControlResMultipler = 0.25f 控制技能递减系数

nControlResNum = 0 控制技能效果递减堆叠层数

fControlResResetTime = 5f控制技能递减重置时间

Buff

包括

大部分Buff单独计算

每一种Buff/Debuff都有自己的ID

Debuff

包括：

List<float> 流血，（物理）持续伤害，物理增伤

List<float> 击倒，（任意）躺在地上

List<float> 浮空，（任意）飞在空中，上升之后再自由落体

List<float> 眩晕，（物理）眩晕，停止行为

List<float> 减速，（任意）减速，List叠加，乘法叠加\*

List<float> 定身，（任意）不能移动

燃烧，（火焰）持续伤害

寒冷，（寒冰）减速，List叠加，乘法叠加\*

冻结，（寒冰）停止行为

中毒，（毒药）减速，List叠加，乘法叠加\*

麻痹，（闪电）停止行为

恐惧，（任意）四处乱跑且不会攻击

瘫痪，（任意）受到任意伤害会被惊醒

缴械，（任意）持续时间内无法攻击，克制物理攻击类型敌人，但是可以施法

沉默，（任意）持续时间内无法施法，克制法师型敌人，敌人的技能将表明是否属于施法

State

防御状态（格挡时，敌人架盾状态，有些技能对防御状态的敌人造成额外效果）

狂暴状态（狂暴期间，敌人狂暴期间，有些技能对进攻状态敌人造成额外效果，如冷静可以结束处于狂暴的敌人）

有些技能和物品的效果针对防御状态/狂暴状态有特殊效果

玩家中只有Warrior可以进入防御状态

玩家中只有Barbarian可以进入狂暴状态

作用：Debuff用于在特定时机做判断 举个例子：A技能击中带有冻结效果的目标会造成额外伤害

获得Debuff时Int加1，失去Debuff时Int减1，驱散效果时重置为0并清除List

实现方法：

Debuff可以叠加层数，伤害可以独立计算

补充Buff/Debuff相关逻辑

Buff/Debuff造成治疗或者伤害

Debuff

命令模式：

新建一个EffectInstance，如果这个EffectInstance存在，就会持续造成伤害。

如果EffectInstace被驱散，或者时间到了之后，就无法再造成伤害。

**怪物逻辑**

怪物模板类型：

\*Animation Speed

windmill旋风斩 0.8

蓄力准备阶段 0.5

**JSON解析**

JSONParse()