# KBE服务端技能详细设计文档(python)

之前的文档是总体设计，最近完成了主动技能的编码。这里以一个丢火球和持续加血buffer为例详细说明编码过程。（以下编码方式不一定是最合适和合理的，如有不当之处请及时提出，谢谢！）

首先我们需要一个技能数据库 路径为：scripts\data\d\_skills.py

每个技能对应一个字典的项， 技能相关的属性和数据都在里面有对应字段，里面有个script字段对应的是这个技能的脚本。

当avator建立时，我们需要有个类专门管理技能相关的内容，所以让avator继承于SkillMananger

class Avatar(KBEngine.Entity,

GameObject,

Flags,

State,

Motion,

SkillManager,

Combat,

Teleport,

Dialog):

SkillMananger中的skillDic字典记录的是所有技能脚本（script字段对应的类实例）

当玩家申请释放技能时，会调用到SkillMananger中的def spellTarget(self, skillID, targetID):

在spellTarget中我的处理是 判断skillDic中有没有skillID对应的技能类实例，然后调用skill实例的skill.spellSkillRequest(self, targetID) 将所有的判断释放技能合法性和过程都在具体类实例里面判断。

技能类的设计：（运用到了模板设计模式）

SkillBase为技能的基类，派生SkillTargetBase 基类，这个基类是有明确目标的基类， 火球类SkillFireBall则继承于SkillTargetBase类， 持续加血buffer继承于SkillTargetBase类，target固定为自己。

添加自己的技能类

我们要添加一个类的话，可以继承于SkillTargetBase类，然后重写onSkillCastOver\_函数，和receive函数。

onSkillCastOver中我们处理的是技能消耗，例如释放技能后会消耗什么，不重写默认是消耗MP。

receive中我们处理的是技能效果。 例如：火球是让目标受到伤害：receiver.recvDamage

而加血buffer是 ：caster.addDBuff(1, self.onBufferTick)

def onBufferTick(self, delTime):

#ERROR\_MSG("skill::onBufferTick")

self.nowTime +=delTime

if(self.nowTime>=self.verTime):

self.nowTime = 0.0

self.caster.addHP(1)

self.allTime -= self.verTime

if self.allTime<=0.0:

self.caster.removeDBuff(1)

我们看skillTargetbase：核心过程

def spellSkillRequest(self, caster, targetID):

"""

virtual method.

技能释放请求

"""

target = KBEngine.entities.get(targetID)

skillID = self.skillitem[0]

if target is None:

ERROR\_MSG("Spell::spellTarget(%i):targetID=%i not found" % (caster.id, targetID))

caster.client.onSpellSkillResponse(GlobalConst.GC\_SKILL\_ENTITY\_NOT\_EXIST, skillID)

return

scobject = SCObject.createSCEntity(target)

self.scobject = scobject

ret = self.**canUse**(caster, scobject)

if ret != GlobalConst.GC\_OK:

ERROR\_MSG("Spell::spellTarget(%i): cannot spell skillID=%i, targetID=%i, code=%i" % (caster.id, self.skillitem[0], targetID, ret))

caster.client.**onSpellSkillResponse**(ret, skillID)

return

self.setCD()

**caster.skillCDDic[skillID] = self.skillitem[5]**

caster.allClients.**onSpellSkillResponse**(GlobalConst.GC\_OK, skillID)

#施法前摇定时

spellTime = self.getspelltime()

if(spellTime<0.05):

self.**use**(caster, scobject)

return

caster.skillSpellDic.clear()

caster.skillSpellDic[skillID] = spellTime

首先判断技能合理性，然后是施法前摇， 同时CD计时，当施法前摇完后会调用use使用技能。

使用技能会调用到：

def cast(self, caster, scObject):

"""

virtual method.

施放技能

"""

delay = self.distToDelay(caster, scObject)

type = self.skillitem[16]

#INFO\_MSG("%i cast skill[%i] delay=%s." % (caster.id, self.id, delay))

if delay <= 0.1:

self.onArrived(caster, scObject)

else:

#INFO\_MSG("%i add castSkill:%i. delay=%s." % (caster.id, self.id, delay))

caster.addCastSkill(self, scObject, delay)

caster.generateFireLight(type, scObject.getID(), self.skillitem[0], delay)

self.onSkillCastOver\_(caster, scObject)

首先计算技能飞行时间，如果飞行时间过短则直接onArrived打到目标。

FirstOne

2015.04.16