新版太阳神三国杀技能卡型视为技

AI设计概述

适用版本：V2-愚人版（版本号：20150401）清明补丁（版本号：20150405）

说明：

本文是对2015年12月26日答百度太阳神三国杀Lua吧吧友问时所作的回信内容的改写。

本文假定读者已熟悉太阳神三国杀Lua武将扩展的有关概念和方法。

本文内容只针对技能卡型视为技AI，稍微会涉及视为卡牌AI，而与其他方面无关。

正文：

关于设计编写视为技AI的问题，我是这样理解的：

我们的最终目的是要让AI代码融入整个太阳神三国杀的AI系统，

那么最好的办法就是先去了解一下太阳神三国杀的AI系统是怎样工作的。

至少要知道，在出牌阶段主动使用卡牌这个环节，AI的工作流程如何。

然后我们就可以知道，其中的哪些内容需要我们自己去填写补充，以及具体应如何进行补全。

太阳神三国杀的AI系统中，主动出牌的入口是smart-ai.lua中的activate函数。

这个函数将完成以下几项工作：

1、更新角色间的敌友关系。

2、更新卡牌保留值等信息。

3、找出所有可以使用的卡牌。

4、对所有可用卡牌根据动态使用优先级进行排序。

5、逐一扫描所有可用卡牌，并根据事先设定好的该类卡牌的使用方式尝试使用此卡牌。

5.1、如果尝试使用卡牌成功，则将具体的使用信息返回给太阳神三国杀，函数结束（此时该角色将真正使用此卡牌）。

5.2、如果尝试使用卡牌失败，则继续进行后续的扫描工作，直至找遍所有可用卡牌。

6、清空可用卡牌记录。

如果这个过程太复杂，那么您只需要知道，

和人类玩家一样，AI系统也主要在考虑两件事情：看看哪些牌能用，以及具体该怎么用。

其中，找出能用的卡牌对应上述过程中的第3步，确定该怎么用对应则第5步。

而其余的步骤，都是在为更好地完成这两件事情做准备罢了。

那么首先来看看这里的第3步：找出所有可用卡牌。

这是通过smart-ai.lua中的getTurnUse函数完成的。

getTurnUse函数将收集本角色的所有手牌（包括木牛流马等区域中视为手牌的牌），

并根据这些手牌产生可用的技能卡。

然后把这些手牌和技能卡根据使用价值从大到小进行统一排序。

接着将依次尝试使用这些手牌或技能卡，确定卡牌是否真的可用。

若可用，则将其加入可用卡牌集合。

最后，getTurnUse函数把这份经过排序的可用卡牌集合作为结果返回给activate函数。

注意这句话：“并根据这些手牌产生可用的技能卡”。

这就和我们开始有关系了。

根据手牌产生技能卡的过程是由smart-ai.lua中的fillSkillCards函数完成的。

这个函数首先会从手牌中排除掉那些不能被直接使用的卡牌

（比如被鸡肋了、或者被“武神”锁定修改了牌面信息的），

然后调出记录着所有AI可用技能信息的sgs.ai\_skills表，依次检查其中所有的技能信息。

这里所说的技能信息其实就是一个table表，它的通常格式为：

{

name = "技能名",

getTurnUseCard = function(self, inclusive)

--技能卡产生函数

end,

}

包含了name和getTurnUseCard两项内容。

其中name是一个string，表示产生某技能卡的技能的名字；

而getTurnUseCard是一个函数，表示该技能在什么条件下、如何产生此技能卡。

这里，参数self为SmartAI表类型，表示当前AI；

参数inclusive为boolean类型，表示是否没有收集到可用的手牌（即，是否只有技能卡可用）。

每当发现一条技能信息的name值恰是本角色拥有的技能时，就会调用这条信息的getTurnUseCard函数产生对应的技能卡。

如果getTurnUseCard函数的返回值是Card类型，那么就认为其产生了对应的技能卡；

而如果返回值是nil，则认为这个技能在此情况下不能或者不应该产生技能卡。

现在回到刚才说的那个第3步，然后注意一下这句话：

“接着将依次尝试使用这些手牌或技能卡”。

在太阳神三国杀的AI系统中，尝试使用某卡牌，主要涉及以下几个函数：

useSkillCard函数（用于尝试使用技能卡）

useBasicCard函数（用于尝试使用基本牌）

useTrickCard函数（用于尝试使用锦囊牌）

useEquipCard函数（用于尝试使用装备牌）

useCardByClassName函数（用于尝试使用特定种类卡牌）

这些都是smart-ai.lua中的函数。

所以在第3步的这个getTurnUse函数中，对于每一个可用的手牌或技能卡，

将根据其基本类型的不同，选用不同的函数进行使用尝试。

对于技能卡呢，选用的自然就是useSkillCard函数了。

useSkillCard函数首先要做的，是形成技能卡代号。

如果是源代码中那些用C++写的技能卡，直接取其类型名（ClassName）作为技能卡代号；

而如果是我们用Lua写的技能卡，

则采用"#"连接其对象名（objectName）的形式作为技能卡代号。

毕竟所有Lua技能卡共用了同一个类型名：LuaSkillCard。

然后useSkillCard将调出记录着各种技能卡使用方法函数的sgs.ai\_skill\_use\_func表，

根据技能卡代号找到具体的使用方法函数，进行尝试使用。

这些使用方法函数的格式为：

function(card, use, self)

--具体的使用方法

end

其中，参数card表示当前要使用的技能卡，

其实就是之前技能信息中getTurnUseCard函数产生的那个Card类型的结果。

参数use表示卡牌使用方式，根据实际情况，

它可能是一个卡牌使用结构体（CardUseStruct），也可能是一个table表。

当然，在当前第3步的这个过程中，它是一个table表。

参数self表示当前AI，与之前getTurnUseCard函数里的那个self是一样的。

sgs.ai\_skill\_use\_func表中的这些使用方法函数的主要任务，就是填充其中的那个use参数。

这里，use.card表示具体要使用的卡牌或技能卡；

use.to表示该卡牌或技能卡的所有使用目标角色。

use.isDummy表示是否为不会造成真正影响的“尝试使用”。

所以，如果使用方法函数结束时，use.card的值不为nil，

就说明使用成功、产生了具体的使用方法。

至此，第3步的主要工作就算完成了，找出了所有可以使用的卡牌。

下面我们接着来看看第5步：尝试使用卡牌。

其实它和第3步的后半部分那次尝试使用卡牌的过程是差不多的，

对于技能卡，依然是先调用useSkillCard函数，形成技能卡代号，

再调用sgs.ai\_skill\_use\_func表中的使用方法函数。

不同的地方在于，在尝试使用之前，会针对卡牌的使用方式再次检查卡牌的可用性

（比如防止被鸡肋）；

而且这一次调用使用方法函数时，

参数use变成了一个卡牌使用结构体（CardUseStruct），而不再是table表了。

因为这一次的使用方法函数结束时，如果use.card不为nil，

就将把这个use参数作为卡牌使用信息传回太阳神三国杀，

也就意味着AI将真正使用一张卡牌或技能卡了，所以必须慎之又慎。

了解了太阳神三国杀AI系统关于主动出牌的工作原理后，

就可以开始写技能卡相关视为技的AI了。

其实就是去填补AI系统的空缺的……

首先要写的就是技能信息，并把它加入sgs.ai\_skills表。

为的是让fillSkillCards函数注意到此视为技的存在，使AI考虑使用我们的技能卡。

如下：

local 技能信息 = {

name = "技能名",

getTurnUseCard = function(self, inclusive)

if 视为技发动条件 then

return 视为技产生的技能卡

end

end,

}

table.insert(sgs.ai\_skills, 技能信息)

还有一种等价的写法：

local 技能信息 = {}

技能信息.name = "技能名"

技能信息.getTurnUseCard = function(self, inclusive)

if 满足视为技发动条件 then

return 视为技产生的技能卡

end

end

table.insert(sgs.ai\_skills, 技能信息)

然后需要写的是技能卡的使用方法函数，并把它加入sgs.ai\_skill\_use\_func表。

为的是让AI知道应如何使用此技能卡，

包括使用什么牌形成技能卡、指定哪些角色为目标等等。

如下：

sgs.ai\_skill\_use\_func[技能卡代号] = function(card, use, self)

if 满足技能卡使用条件 then

use.card = 视为技产生的技能卡

if use.to then

use.to:append(目标角色)

end

end

end

这里解释一下中间的那个“if use.to then”是怎么回事。

我们知道，第3步和第5步中为了尝试使用技能卡，会分别调用一次这里的使用方法函数。

而参数use在这两步中的类型是不同的，

第3步中是table表，第5步中是卡牌使用结构体（CardUseStruct）。

这就意味着，参数use中是有可能不存在表示使用目标的to的

（卡牌使用结构体中当然会有to，但table表里就不好说了）。

所以需要在添加目标角色之前，先判断一下这个参数use中是否有to存在。

这就是出现这句话的原因。

还有一个问题，如何使用视为技产生技能卡？

这里通常用到一个函数：sgs.Card\_Parse(卡牌构成字符串)

首先我们将根据技能名、技能卡名、以及形成技能卡所用的卡牌ID，构造一个“卡牌构成字符串”，

然后调用sgs.Card\_Parse函数，就会得到具体的一张技能卡。

下面来介绍一下卡牌构成字符串的写法。

卡牌构成字符串常见的格式有五种：

1、视为卡牌：objectName:skillName[Suit:Number]=IDs

2、C++技能卡：@ClassName=IDs:UserString

3、Lua技能卡：#objectName:IDs:UserString

4、C++技能卡EX：@ClassName[Suit:Number]=IDs:UserString

5、Lua技能卡EX：#objectName[Suit:Number]:IDs:UserString

而我们只需要关心其中的第3种在Lua中的写法，以及第1种视为卡牌的写法。

如果您需要为您的技能卡加上花色和点数信息，则采用第5种写法。

不过通常技能卡是无花色无点数的，所以这里就先忽略了。

PS：其实Lua写法在形式上只是把C++写法中的"@"改成了"#"、把"="改成了":"而已……

注：所有上述后四种格式中的最后一部分“:UserString”都是可选的，通常不需要写。

这样四种格式就被简化为（注意Lua写法中不能省略最后的冒号）：

1、C++技能卡：@ClassName=IDs

2、Lua技能卡：#objectName:IDs:

3、C++技能卡EX：@ClassName[Suit:Number]=IDs

4、Lua技能卡EX：#objectName[Suit:Number]:IDs:

但如果在使用sgs.CreateViewAsSkill函数创建视为技时，view\_as部分出现了setUserString函数，

就不能省略UserString部分了，具体内容请根据视为技实际情况决定。

所以，对于Lua技能卡，在不考虑UserString的情况下，

我们只需要知道它的对象名（objectName），以及所有构成它的卡牌的ID，

即可写出其对应的卡牌构成字符串。

比如要使用卡牌card1和card2产生一张名为MySkillCard的技能卡，那么具体的写法就是：

local card\_str = "#MySkillCard:"..card1:getEffectiveId().."+"..card2:getEffectiveId()..":"

或者更直观一些，使用下面这种等价的写法：

local card\_str = string.format("#MySkillCard:%d+%d:", card1:getEffectiveId(), card2:getEffectiveId())

注意多张子卡ID之间用加号"+"连接。

如果是无需卡牌便可生成的技能卡，那么IDs部分要用点号"."填充，表示此技能卡没有子卡。

比如直接产生一张无子卡的技能卡MySkillCard，写法如下：

local card\_str = "#MySkillCard:.:"

这种写法在不关心子卡构成的情况下也是可以用的

（比如技能信息中的getTurnUseCard函数部分）。

对于视为卡牌，除了要知道其对象名、确定所有构成其的卡牌ID外，

还需要确定视为卡牌的花色和点数，以及技能名。

比如，发动技能MySkill将卡牌card1当做一张花色为红心、点数为7的【桃】使用，

对应的卡牌构成字符串就是：

local card\_str = "peach:MySkill[heart:7]="..card1:getEffectiveId()

或者：

local card\_str = string.format("peach:MySkill[heart:7]=%d", card1:getEffectiveId())

同样，多张子卡ID之间用加号"+"连接，无子卡时用点号"."补充IDs部分。

所以通常会在AI代码中看到类似如下内容：

local card\_str = string.format(

"slash:MySkill[%s:%s]=%d",

card1:getSuitString(),

card1:getNumberString(),

card1:getEffectiveId()

)

这就是发动技能MySkill将卡牌card1当做同花色同点数的【杀】使用的卡牌构成字符串。

按上述方法填好sgs.ai\_skills表和sgs.ai\_skill\_use\_func表之后，

AI系统就可以真正使用我们的技能卡了。

不过为了让AI系统更好地使用技能卡，有时我们会做一些其它的工作。

包括提供此技能卡的使用价值和使用优先级、通过此技能卡的使用情况进行身份判断，等等。

【注意：这些工作不是必需的。】

如果您只想让AI能够使出我们的技能卡而不关心其它问题，那么后面的内容可以略过了。

我们通常会在写好技能卡的AI后顺手填写sgs.ai\_use\_value表和sgs.ai\_use\_priority表，

以提升此技能卡在第3步进行统一排序时的排位，达到优先使用技能卡的目的。

如果不填写，那么AI将优先考虑其它卡牌，

于是您会发现我们的技能卡总是在出牌阶段最后才被使用的那一个。

卡牌使用价值表sgs.ai\_use\_value表记录了一张卡牌或技能卡的使用价值。

填写方法是：

sgs.ai\_use\_value[ClassName] = 使用价值 --卡牌或C++技能卡

sgs.ai\_use\_value[objectName] = 使用价值 --Lua技能卡

注意这里要用到的只是Lua技能卡的对象名（objectName），不需要在前面添上"#"。

比如，一张名为MySkillCard的技能卡，其使用价值是3.7，具体写法就是：

sgs.ai\_use\_value["MySkillCard"] = 3.7

卡牌使用优先级表sgs.ai\_use\_priority表记录了一张卡牌或技能卡的使用优先级。

这个优先级是相对静态的，与第4步中提到的“动态使用优先级”稍有不同。

主要表现在，动态使用优先级会根据卡牌的花色、同类卡牌的数量多少等因素

对优先级数值进行调整，是一种与游戏环境相关的优先级。

卡牌使用优先级表的填写方法与使用价值表类似：

sgs.ai\_use\_priority[ClassName] = 使用优先级 --卡牌或C++技能卡

sgs.ai\_use\_priority[objectName] = 使用优先级 --Lua技能卡

比如，一张名为MySkillCard的技能卡，其使用优先级是8，具体写法是：

sgs.ai\_use\_priority["MySkillCard"] = 8

除此之外还有一个卡牌保留价值表sgs.ai\_keep\_value，记录了一张卡牌的保留价值，

主要在第2步起作用。

不过由于技能卡并不是真正的卡牌，所以也不需要考虑其保留价值，此表不需要填写。

卡牌保留价值表的填写方法只有一种：

sgs.ai\_keep\_value[ClassName] = 保留价值

有时我们会需要AI刻意去保留某些种类的卡牌，为技能卡的使用做准备。

比如技能“强袭”需要武器牌，技能“银铃”需要黑色牌等等。

那么可以按一定的格式构造特定技能保留价值表或特定技能花色价值表，

使AI留意我们需要的卡牌。

特定技能保留价值表的格式为：

sgs.技能名\_keep\_value = {}

表中可以为某种卡牌类型指定其保留价值。

比如，技能MySkill需要用【杀】或武器牌发动，就可以这样写：

sgs.MySkill\_keep\_value = {

Slash = 6,

Weapon = 6,

}

特定技能花色价值表的格式为：

sgs.技能名\_suit\_value = {}

表中可以为某种花色代号指定其保留价值。

比如，技能MySkill需要用草花牌发动，就可以这样写：

sgs.MySkill\_suit\_value = {

club = 7,

}

为了更好地与队友配合，AI系统还提供了一个卡牌需求表sgs.ai\_cardneed表。

当填写了sgs.ai\_cardneed表之后，

那些拥有分配卡牌能力的队友就会考虑在分牌时，先将您需要的卡牌交给您。

卡牌需求表中记录的是一个个具体的卡牌需求判定函数，函数的格式为：

function(friend, hcard, self)

--判断队友friend是否需要卡牌hcard的过程

end

注意这个函数是站在卡牌分配者的角度思考问题的，

所以参数friend才是拥有我们技能的角色，

而参数self表示的是卡牌分配者的AI，

参数hcard表示在卡牌分配者手中等待分配给friend的那张卡牌。

如果这个函数的结果是true，那么卡牌分配者就会考虑将hcard交给friend。

将卡牌需求判定函数写入卡牌需求表的格式为：

sgs.ai\_cardneed[技能名] = 卡牌需求判定函数

比如，技能MySkill需要用锦囊牌发动，就可以写为：

sgs.ai\_cardneed["MySkill"] = function(friend, hcard, self)

return hcard:isKindOf("TrickCard")

end

某些技能卡带有明显的敌对或友好倾向，为了能让AI系统认识到这一点，以便更好地判断场上身份，

AI系统提供了一个卡牌使用仇恨值表sgs.ai\_card\_intention表。

这个表记录了有关某种特定卡牌或技能卡的情感态度信息。

填写方法有四种：

sgs.ai\_card\_intention[ClassName] = 仇恨值 --卡牌或C++技能卡

sgs.ai\_card\_intention[objectName] = 仇恨值 --Lua技能卡

sgs.ai\_card\_intention[ClassName] = 仇恨值更新函数 --卡牌或C++技能卡

sgs.ai\_card\_intention[objectName] = 仇恨值更新函数 --Lua技能卡

前两种适用于技能卡仇恨值完全固定的情况，

只要对目标角色使用了此技能卡，就一定表示某种特定的情感倾向。

其中，仇恨值为正数时，表示敌对；仇恨值为负数时，表示友好。

比如，一张名为MySkillCard的技能卡的仇恨值是-80，写为：

sgs.ai\_card\_intention["MySkillCard"] = -80

这就意味着，在游戏过程中，只要对主公使用了此技能卡，就会被AI认为是在表忠

（因为负数仇恨值表示示好）。

而对大家公认的反贼使用此技能卡，则会被认为是在跳反。

后两种适用于技能卡仇恨值复杂，或者对多个目标角色应区别对待的情形。

其中的仇恨值更新函数是一个用于具体讨论使用者情感倾向的函数，其格式为：

function(self, card, from, tos)

--具体的讨论过程

end

在这里，参数self表示观察者AI（通常是最先加入游戏的一号位角色），

参数card表示所使用的卡牌或技能卡，

参数from表示卡牌或技能卡的使用者，

参数tos是table表类型，表示卡牌或技能卡的所有目标角色。

比如，某张名为MySkillCard的技能卡，对使用者攻击范围内的角色表示敌对，

对其他角色表示友好，就可以这样写：

sgs.ai\_card\_intention["MySkillCard"] = function(self, card, from, tos)

for \_,to in ipairs(tos) do --逐一扫描所有目标角色

if from:inMyAttackRange(to) then --如果当前目标在使用者攻击范围内

sgs.updateIntention(from, to, 80) --表示敌对

else 否则

sgs.updateIntention(from, to, -80) --表示友好

end

end

end

代码中出现了一个很关键的函数：sgs.updateIntention(源角色, 目标角色, 仇恨值)

这个函数用于更新一名角色对另一名角色的仇恨值，是非常常用也非常方便的一个函数。

由于仇恨值更新函数并没有返回值，所以要想表达某种情感倾向，就必须在函数内部完成仇恨值更新工作，

这就是为什么在很多AI文件中，仇恨值更新函数经常和sgs.updateIntention函数同时出现的原因。

以上就是对技能卡型视为技AI的设计编写问题的理解，希望能够有所帮助。

谢谢阅读！

By：独孤安河