CNCSAGE 引擎和 META MOD 的 LUA 代码教程

作者：Mjjstral aka. MetaIdea (Maximilian F.)
译注：星火战队-Ngen

摘要：本文是 CNCNZ 论坛的 Mjjstral aka. MetaIdea (Maximilian F.)制作的 MetaMod2.00 的自述文件中 Lua 教程的译注文。MetaMod 是凯恩之怒的一个 Mod，这个 Mod 最大的特点就是使用了大量先进而复杂的 Lua 代码，实现了游戏内实时编辑运行 Lua 代码、读写地图内具体单位标志信息、改编存档等神奇游戏效果，学习这个 Mod 的代码对于研究 Lua 在 RA3Mod 中的作用有重大意义。原文的前半部分主要介绍了 KWMod 的安装和 MetaMod 内置防御/塔防地图的玩法，后半部分则是 Sage 引擎中 Lua 代码的教程，后半部分对与研究 Lua 很有帮助，这里我对后半部分进行了翻译并注评，评注主要是指出在 RA3 的环境下相关代码与 KW 的区别。希望得到这个 Mod 及其源码请下载附件，这个 Mod 本身是开源的而且鼓励改编。实际上 MetaMod 本身也有开发工具的性质，因为它实现了在游戏内实时编辑运行 Lua 代码这个重要功能，即不需要反复打包 Mod、开关游戏就能实现 Mod 中 Lua 代码的改动，这给 modder 带来了极大的便利，因而这个教程中有很多部分是在介绍如何使用 MetaMod 作为 Mod 的开发工具。只可惜这个 Mod 不能直接用在 RA3 中，只关心 Lua 在 RA3ModSDK 中实用功能的读者在读到此类时可以选择略读或者略过，跳过 2、6、7、8，只看 1、3、4、5、9，而其中 5、9 最为重要。当然如果你有能力可以尝试将游戏内实时编辑运行 Lua 代码的功能复刻至 RA3 中。为了保证代码本身保留字、函数名、xml 模块等词的可搜索性，这些词选择不译，另外译者并非 Lua 程序员，如果相关专业术语翻译错误请多多包涵。

一、Lua 一般信息

《命令与征服 3：泰伯利亚战争》和《凯恩之怒》使用的所有 Lua 函数的语法和类型词都是 4.0.1 版，并且它们的参数使用官方的 Lua4 引用方法(参考 refman-4.0.pdf) 。要查找语法错误可使用 luac4.exe，这在 metamod 的压缩包里有。比如一段错码会这样提示：

```
@echo off

luac4.exe

"C:\LUA_INPUT.lua" pause
```

你可以选择 933 种不同的游戏内 Lua 代码指令来创造出非常复杂的游戏机制。Lua 本身提供了 95 个内部 Lua 函数。另外在 Meta Mod2.0 里有大量新奇而且强大的函数。

下面这个重要的列表是在 Meta Mod 压缩包里找到的：ALL_SAGE_LUA_COMMANDS.txt

这是 CNC3 的 Sage 引擎内部可行的所有 Lua 函数、指令、变量的列表，同时也包含了 Lua 本身的函数语法和一个 LuaEvent 代码表。这个列表包含了许多在 scripts.ini 里找不到的 ExecuteAction 和 EvaluateCondition 指令。这也许也能在同样出自 EALA 的《魔戒：中土之战》系列游戏中有用。至于如何给这些 CNC Sage 函数配用正确的参数请看下面的教程。更多关于如何使用 Lua 本身的函数的信息请看 refman-4.0.pdf。

另一个文件 METAMOD_LUA_REFERENCE_LIST.txt 是关于 Mjjstral 的 Meta Mod2.0 中大部分函数、变量、表的列表。要使用的话请参考实际源码以详细了解参数的类型。在很多情况下函数的自变量是非必要的，你可以用 nil 值或直接不填来跳过某些自变量，根据上下文可知这些自变量是 string 还是 table 或者 number 类型。

二、快速写、测 Lua 代码的方法

用参数-win 打开 KW。打开 LIVE_INPUT.lua 文件和 LIVE_OUTPUT.txt 文件然后让文本编辑器放在游戏窗口旁边。然后使用 meta control 建造栏中的“Run Lua Script”按钮或者使用 Force_Lua_Script_Button = yes 来让协议技能栏出现“Run Lua Script”按钮。然后你可以直接在 LIVE_INPUT.lua 里输入代码，也可以重写已有函数。查看 ErrorLog.txt 来找错。我也推荐使用 Dev_Mode = yes。要输出任何值到 LIVE_OUTPUT.txt 就使用 out(yourtext)函数，而用 print(yourtext)函数则是把值显示到游戏里。

三、下列行为在 Lua 代码中是可能的

1. 在执行代码中用 string 类型参数（在 ObjectType 变量中也行）
   注 1: 所谓执行代码 (action script) 就是指真正产生实际游戏效果的 933 个游戏内部 Lua 函数。

2. 在执行代码中用 real 或 float 类型参数

3. 使用下拉列表的参数（通过用 n-1 值，n 为该参数在下拉列表的序数）
   注 2: 在地编脚本编辑器里具体脚本的参数选择时，有些参数是通过在下拉列表里选择来使用的，这里简单说一下 Lua 函数怎么引用地编下拉列表的值。有两种方法: 1.直接用字符串，下拉列表里写着啥就照抄进 Lua 函数，然后用引号括起来。2.根据这个参数在下拉列表中的序数（第几个），把序数-1 作为数值写进 Lua 里。

4. 在执行代码中用队伍名参数

5. 使用布尔值（0 或 1）

6. 文件的写入导出和各种操作（例如保存游戏，自动用 config 参数打开游戏，引用外部 mod 代码）

7. 播放自定义视频，用雷达况播视频，播 Sound 和 Music

8. 单位的属性变化以及任意方式 storage

9. 用 EvaluateCondition 系列函数得到单位的信息：如 status、kindof、objecttype、数量等等

10. 自定义胜利条件

11. 计数器和计时器

12. 规划代码用的代码计时器

四、Lua 代码的限制：

1. 有些执行代码不能用 integer（整数）类型参数（只是少量）

2. 不能用 player 参数：不过大部分执行代码能用队伍名参数代替，所以我们可以靠它。在本文中提到的函数都能用。

3. 执行代码不能直接用地图绝对坐标系的值，只有 SpawnAtPosition(...)这个函数能用。

五、在 CNC Sage 引擎中 Lua 函数的参数使用规则：

关于 CNC Sage 引擎中非 ExecuteAction 和 EvaluateCondition 类函的 Lua 函数：这些函数只会引用 table 类型的 Object 参数（不像 ExecuteAction 类函数），所以一定要用 GetObj.Table(Object)函数。注意在 globals() table 中有但没有在 Lua 注册过的 Object 不能被这些函数直接使用。

注 3: 所谓“在 globals() table 中有”就是说这个 GameObject 的本级（与 kindof 同级）属性值 ForceLuaRegistration="True"或者这是在地编里放置的物体。而“在 Lua 注册”是指在 AIUpdateModuleData 下的 AILuaEventsList 有相关的 EventList，不过这只是对 KW 而言，对于 RA3 并不能在 AI 段注册，而要在 Behavior 段用<LUAEventList>注册。

Object 参数的类型：

- Table 类型参数：
  这些 Object 存储在 lua 的 globals()这个 table 中（所有的 lua 函数和 global 变量也是）并且它们也都是 table 类型的值。这些 Object 的 table 类型值都有一个独特的以“OBJID”开头的标志值，所以我们可以容易地滤选它们。所有在 GameObject 的 xml 中的 AIUpdate 模块中有 AILuaEventsList 描述的 Object 都会在 globals()中有一个相应的 table 类型值。你可以在 xml 里用 ForceLuaRegistration="True"来强行注册。
  注 4: 同注 3
- String 类型参数：
  所有 Object 都有多种和它们有关的 string 类型名字。这类的引用值可以在所有使用某 Object 作为参数的 ExecuteAction 和 EvaluateCondition 指令中用到。

其他参数类别：

- 队伍名（Team names）：
  使用 GetTeamName(Object)，可以得到某单位的队伍名。
  在 MetaMod 中队伍名取值范围是：ClientTeamName, NeutralTeam, CreepsTeam, CivilianTeam, ObserverTeam。
  在 MetaMod 中含队伍名的 table 是：GlobalTeamTable, HumanPlayerTable, AIPlayerTable, AllTeamsTable。
  需要实际队伍名值请看 MetaMod 的源码(scripts.lua)。
- 路径点在这两个 table 中：WaypointTable, WaypointDistanceTable
- 区域名：在 AreaTable 中(未过滤)
- 布尔值（TRUE/FALSE）：用 0 或者“0”代表否，用 1 或者“1”代表是。
- 下拉列表参数（通过使用 n-1 值，n 为该参数在下拉列表中的序数）：如果在地编里选择参数时有一个下拉列表，那么你就需要在相应的 Lua 中使用 n-1 作为数值或者 string 类型值。大部分下拉列表在 MetaMod 中已经用 table 引用了，直接使用 table 里的值就行。
- String 类型值（字符串）：在地编里能用文本框任意改的值在 Lua 中都可以使用 string 类型值作为参数。
- Real 或者 Double 类型值（实数或双精度实数）：在 Lua 中你可以使用数字甚至带数字的字符串来表示 real 或者 double 类型值。
- Integer 类型值（整数）：除了与之相关的运算符之外似乎没有别的方法能使用整数。当读取进入电脑端时，Lua 的电脑-代码接口会把所有在 Lua 代码中提及的数字都被完全地转化为 double 类型。
- 坐标系类型值：似乎 Lua 中不能用。
- Player 类型值：在已知方法中没有一种能直接使用 Player 类型参数。从内部看 Player 类型参数引用时似乎是 struct 类型（结构体）。实际上 Player 类型参数由代码文件夹名和 Player 名组成，即是"WBScriptFolder/Playername"形式，举个例子："Player_1/<This Player>"。好消息是在大多数执行代码中你可以用相应的 team 类型值代替。另一个能间接使用 Player 类型值的地方是是用 SendScriptHostCodeMessage 的方法法，这会在 2.7 地图整合指南那部分说。

ExecuteAction 和 EvaluateCondition 系列函数：

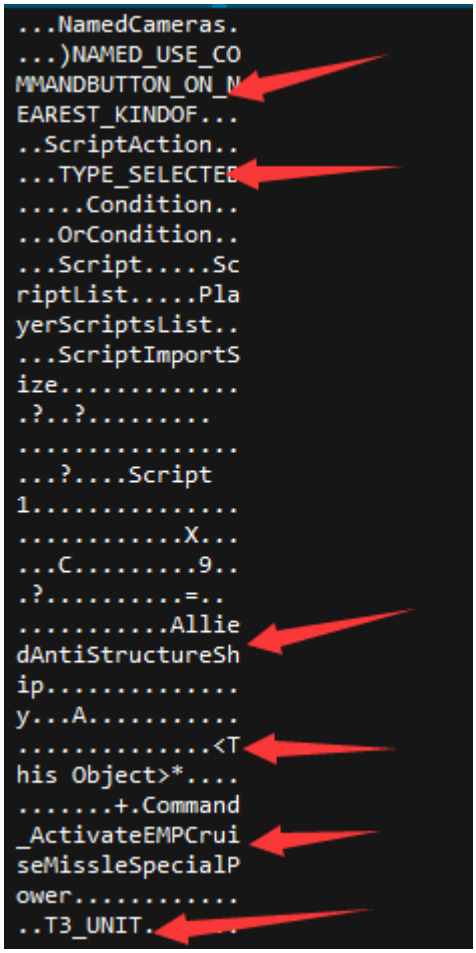
这些是可用函数中最最有价值的。所有牛逼玩意都能用这些函数实现。广泛地使用这些函数是 MetaMod 的主要特点。ExecuteAction 系列函数能让你使用大部分地编脚本功能，而 EvaluateCondition 系列函数则让你能使用大部分地编脚本的 IF 条件。

破游共包含 677 个 ExecuteAction 函数以及 218 个 EvaluateCondition 函数，这些你都能尝试使用。你可以在 MetaMod 的下载包里找到这些函数和所有内部指令的完整列表。

至于如何给这些函数找合适的参数：
- 用 MetaMod 文件包里的 ALL_SAGE_LUA_COMMANDS.txt 列表，得到第一个参数。
- 也可以在 scripts.ini 中得到第一个参数，但有些参数在那是没有的。
- 使用地编的脚本编辑器菜单来查看所有参数的及其顺序。你可以输出一个脚本 scb 文件然后用编辑器打开来查看第一个参数，通常是一段用大写字母和下划线写的东西。你只需要加上引号就能在 Lua 里使用这个参数。

- ExecuteAction:
  语法：ExecuteAction(command,parameter1,parameter2,...)
  引用某 Object 时：既能用 string 类型值也能用 table 类型值
- EvaluateCondition:
  语法：EvaluateCondition(command,parameter1;parameter2,...)
  引用某 Object 时：只能用 string 类型值，所以用 GetObj.String 函数以确保正确的值类型。
  相关的运算符：使用“n-1”（n 为运算符的序数）或者使用 CompareTable["operator"]
  (运算符顺序：>, >= ,< ,<=, ==, ~=)

注 5：这里原文说得比较模糊，且 RA3 与 KW 地编区别比较大，很多有用的函数在 ALL_SAGE_LUA_COMMANDS.txt 里并不能找到，而且似乎 RA3 也没有 scripts.ini 文件，我简单说一下怎么利用 RA3 的地编中的脚本编辑器查找 Lua 函数的相关参数。首先打开地编脚本编辑器，编辑一段你喜欢的脚本，然后输出脚本（Export scripts），生成一个 scb 文件。用记事本或者 UE 打开这个文件，会发现一大堆乱码，但仔细搜索还是能发现几句人话。



例如这是“当选择航母时，航母对最近的 T3 单位使用 EMP”这个脚本，可以看到 NAMED_USE_COMMANDBUTTON_ON_NEAREST_KIND 就是对最近单位用某个 cmd 的意思，这可以作为 ExecuteAction 函数的第一个参数写入 Lua 里，然后<This Object>是第二个参数，不过通常我们会用 self 来作为这类指涉单位的参数，以指涉在主函数里引用的【这个单位】，而 Command_ActivateEMPCruiseMissleSpecialPower 则是第三个参数，T3_UNIT 就是第四个参数，因此这句可以写为 ExecuteAction(“NAMED_USE_COMMANDBUTTON_ON_NEAREST_KIND”,self, “Command_ActivateEMPCruiseMissleSpecialPower”,”T3_UNIT”)

而它的条件则是一个 EvaluateCondition 函数，这个函数的第一个参数是 TYPE_SELECTED，第二个参数是 AlliedAntiStructureShip，于是整个函数写为
EvaluateCondition(“TYPE_SELECTED”,”AlliedAntiStructureShip”)
所以整段脚本在 Lua 中的表达就是：
If EvaluateCondition(“TYPE_SELECTED”,”AlliedAntiStructureShip”)==1
Then ExecuteAction(“NAMED_USE_COMMANDBUTTON_ON_NEAREST_KIND”,self,
“Command_ActivateEMPCruiseMissleSpecialPower”,”T3_UNIT”)
End
可以发现，这里面的参数顺序不一，脚本条件和执行的动作混杂在一起，怎么在其中区分出正确的参数还是要很考验各人的灵视的。

六．MetaMod 中一些特别函数的信息

包含超过 10000 行的 lua 代码的 Meta Mod 2.00 提供了很多有用的函数并且使 modder 有可能拥有超越地编脚本编辑器的能力。在数以百计的函数、表、变量中，最为重要的应该是这些：

FUNCTION	COMMENT
SetScriptTimer(time,action,loops,condition)	in 1s steps, loops=-1 for infinite, action is a function or string
SetScriptTimerFast(time,action,loops)	in 0.01s steps
SetDelayed(func)	shortcut for ScriptTimerFast with 0.01 delay
spawn(ObjectType,team,number,ref,waypoint,ActionOnObject,OnObjectType)	
SpawnAtPosition(ObjectType,team,x,y,z,ObjectRef,orientation,Waypoint,numbercount,fast,ActionOnObject)	fast: 0=normal,1=fast,2=superfast
GetObjectPosition(Object)	returns a table, accessing coordinates by table.x and table.y
GetBaseRef(team)	returns an object reference of the original base hq
push(Task)	push tasks on the TaskStack that depend
	on games simulation time,Task can be a function or a string
pop(Task)	every Task pushed should contain pop()
GetTeamName(teamobject)	
GetTeamsDescriptiveName(input)	input can be team or object
GetFactionName(input)	input can be team or object
GetStringRefListOfAllObjects(CreepsTeamToo)	
GetStringRefListOfAllObjectsForTeam(team)	
print(output, display_time)	

WriteToFile(output,file,mode,hide) or out(...)	If no file is specified, LIVE_OUTPUT.txt will be used
RandomString(l)	l=length
SetCounter(counter, value, text, no_display)	
SetTimerCounter(value, text, counter)	
delete(thisobject)	
FireWeaponOnObject(Object, Weapon)	
FireWeaponPosition(Weapon,x,y,waypoint,OnObjectType)	
flash(input)	team or object
GiveChoosePositionOption(team,action,action_with_position,waypoint)	
GiveAcceptDenyOptions(team,acceptaction,denyaction,preaction)	
GPOT	GlobalPlayerOptionTable: Use GPOT[teamname][attributename] important attributenames: "teamname","name", "faction", "ai"
GlobalTeamTable, HumanPlayerTable, AIPlayerTable	
ClientTeamName, NeutralTeam, CivilianTeam, CreepsTeam, ObserverTeam	
Human_Players_Count, AI_Players_Count	

注意：使用 Meta Mod 的源码以了解细节！需要完整列表请看 METAMOD_LUA_REFERENCE_LIST.txt 文件。在很多情况下函数的自变量是非强制的，考虑到具体情况，大部分自变量是 string 或者 table 或者 number 类型。你可以用 nil 值或简单地不填来跳过某些自变量。

产生单位以及位置信息：

SpawnAtPosition 函数不能没有延时。延时时长取决于代码原理！所有用这种方法产生单位的执行代码都需要用代码计时器延时或者 push(Task)或者把"ActionOnObject"作为最后一个变量。

如果你想产生一个新的带有 IMMOBILE 这个 kindof 的 mod 单位，你就需要新加一个 spawnextender 入口（1.在 xml 里看 Object_MetaControlDummyOCLSpawnExtender.xml. 2.在 lua 的 OCLSpawnExtenderTable 这个 table 里写代码。否则产生这个单位时游戏就会炸掉。

用 spawn(...)就不会有有以上两个缺点，但你只能在 0,0 坐标点用 objecttype 的方式产生单位，然后使用路径点。

七、地图整合指南

有三类地图整合方式：

1.MetaMod 可以和加了特定地编脚本指令的地图联系。如果只想让外交联盟变化的选项正常运行，把 METAMOD_MAP_SCRIPTS_CUSTOMMAPS.scb 脚本文件导入你的地图就行。

2.要把 mod 自带的防御/塔防玩法搞得更好，你可以给出确切的路径点名、路径名以及单位名。这些在防御图中不是强制的但能帮助 MetaMod 的算法使用更好的单位产生点和行走路径。至于塔防地图，必须给出确切的塔名以让 MetaMod 能检测和工作。

通过这些手段，你可以做一个有 MetaMod 内置防守玩法之先进出兵机制的新防守或者塔防地图。这意味着你再也不需要考虑出兵代码了。如果你喜欢改变 MetaMod 的出兵行为，你可以用外部的 mod 代码重写那些代表性的函数或者直接把代码加到 autostart.lua 文件中。这意味着你可以给你的地图配上额外的防御/塔防地图的 mod 代码，通过改写 MetaMod 的一般函数抑或只是加上了仅会在特定地图使用的代码。

请使用下列名称约定以便 MetaMod 和地图整合：

对于防守或者塔防图：

1.创建路径点：

路径点名: METAMOD_NAOD_SPAWN_WAYPOINT (强制使用 MetaMod 的玩法)

另一类路径点名: METAMOD_NAOD_SPAWN_WAYPOINT_OPTIONAL (只有当用户启动时才使用 MetaMod 的玩法)

2.创建可选的路径：

路径名: METAMOD_NAOD_WAYPOINT_PATH

第一个路径点名: METAMOD_NAOD_WAYPOINT_PATH_START

对于塔防：

塔名：METAMOD_TOWERDEFENSE_OBJECT

有了以上的约定，MetaMod 会自动地检测到路径点或者塔，然后开始它自带的防御/塔防玩法。你也可以用自己的产生单位脚本，只要"metamod_modstatus_counter"值不是 0，就像玩家没开 MetaMod 一样。

3.LUA 中"用户端-代码执行端"的联系桥

在 MetaMod 地图整合中有一种特殊情况出现在了所有 KW 官方图。这些地图因为全球征服模式而全都引用了一个通用代码。而你可以用这个引用来把你自己的代码一次性插入到所有地图中，只要编辑这个地图中的代码：
libraries/lib_mg_tacticalplay_skirmish/lib_mg_tacticalplay_skirmish.map

注意到在这些通用代码中的计数器不会和在 Lua 环境下使用的计数器共享，所以我们需要借由 MetaMod 的 SendScriptHostCodeMessage(code,counter)函数来把它们联系起来。这只不过是简单地把计数器值从用户端经过一个虚单位的经验转化到了在“代码执行“端的一个计数器上。

八、LUA 输出和输入

MetaMod 使用一个叫 MetaModIO 的工作文件夹，在这：

C:\Users\"USERNAME"\Documents\Command & Conquer 3 Kane's Wrath\MetaModIO

或者在这:

C:\Users\"USERNAME"\AppData\Roaming\Command & Conquer 3 Kane's Wrath\MetaModIO

罕见情况在这:

...\"C&CKane'sWrath GameFolder"\RetailExe\1.2\...

你也可以通过游戏内的 meta command menu 栏-> mod info -> open MetaModIO folder 来打开文件夹。游戏会最小化然后打开相应的文件夹。

- 1. "AUTOSTART_CONFIG.lua":这是被游戏读取的主文件。你可以在其中插入任何 Lua 代码然后它就会在地图开始时执行。你也可以重写 MetaMod 的 Lua 函数、值和表，尽管大部分情况下推荐使用 mod 原本的代码。
- 2. "METAMOD_SAVEDATA":所有元数据，比如等级，xp 点，代币和最高分都储存在这里。注意这个文件是加密的。
- 3. "METAMOD_SAVEDATA_BACKUP"：名字表示备用，是在每次载入一个新地图时第一个存档生成时产生的。
- 4. "LIVE_INPUT.lua"：这玩意对 modder 来说太他妈的有用了，你可以在这里写 Lua 代码然后在游戏里执行，只要你用那个在 meta tab 菜单里的 Lua 代码按钮，或者在 autostart config 里用 Force_Lua_Script_Button = yes 调出一个协议栏技能。如果没这玩意这个 Mod 根本不会做出来。
- 5. "LIVE_OUTPUT.txt"：这是用 WriteToFile(...)函数或者它的简写 out("text")函数时输出时产生的。而且在 meta command 菜单的“other”下大部分选项都会将它们的输出值转到这个文件。你可以输出整个单位列表或者其他有用的玩意来修理 bug 或者测试。
- 6. "templates.txt"：这个文件存贮了你在 meta command 菜单"templates"下你能创造的模板。每个模板都是一个函数，这个函数至少包含单位的 type 和它们的相对位置。
- 7. "ErrorLog.txt":在修理 bug 时非常有用，比如错别字或者语法错误，特别是在你用"LIVE_INPUT.lua"在游戏内允许代码的时候。
- 8. "PreLoad.lua"：比任何 Metamod 函数都先执行的函数。没用过，搞这个只是为了修理 bug。
- 9. mod script files：所有外部文件的运行框架，情看下一节

给 MetaMod 外部 mod 代码运行器写 Lua mod 代码：

所有 Lua 代码必须以"script."开头而以".lua"结尾存储于 MetaModIO 文件夹例如 script.ExampleScript.lua，这样才能被 MetaMod 识别。有关事项请打游戏内 meta command 中的"external scripts"开子菜单。那里也会有创建"ExampleScript"的选项，这将会周期性地把“hello world“打在屏幕上。请以这个代码作为基础来创造你自己的代码。

要创造你自己的代码，首先请在"LIVE_INPUT.lua"的帮助下写入并且测试你的代码。要快速找出语法错误请用上面提到的 luac4.exe 法。

你也可以改写任何在 MetaMod 内存在的函数或者值。我建议你在这里发布你的代码：http://forums.cncnz.com/forum/59-tiberium-warskanes-wrath-maps-modding/ 或者在 moddb 也行。

我会把你的代码打包进未来的 MetaMod 中，如果你发给我的话。

九、与 Lua 相关的 xml 代码

你有几种方式来触发 Lua event（在 scriptevents.xml 里）然后触发 Lua 函数（在 scripts.lua 里）

1. AILuaEventsList (AIUpdateModuleData)：这是给予一个单位触发 Lua 函数的能力的一般方法，通过一个或多个在 scriptevents.xml 中描述的 Lua event。

<AI>

<AIUpdate id="ModuleTag_AI"

AutoAcquireEnemiesWhenIdle="NO"

AILuaEventsList="ExampleEvent"/> </AI>

注 6：在 RA3 中不能在 AI 段注册 Lua，而应该在 Behavior 段注册 Lua，像这样：

<LUAEventList

id="ModuleTag_LUAEventList"

EventListName="ExampleEvent"/>

scriptevents.xml 中 event 里例子是这样的：

<EventList Name="ExampleEvent" Inherit="BaseScriptFunctions">

<EventHandler EventName="OnCreated" ScriptFunctionName="ExampleEvent1_Function" DebugSingleStep="false"/>

<EventHandler EventName="OnDestroyed" ScriptFunctionName="ExampleEvent2_Function" DebugSingleStep="false"/>

</EventList>

你可以像这样根据 modelcondition 或者 objectstatus 自定义 event：

<ModelConditionEvent Name="Selected">

<Conditions>+SELECTED</Conditions>

</ModelConditionEvent> 或者

<ObjectStatusEvent Name="UnderConstruction">

<Conditions>+UNDER_CONSTRUCTION</Conditions>

</ObjectStatusEvent>

2.LuaEventNugget：被这个武器击中的物体会进入一个 LuaEventNugget 里写的的那个 Lua event。例如：

<WeaponTemplate id="ExampleWeapon" Name="ExampleWeapon"

RadiusDamageAffects="ALLIES ENEMIES NEUTRALS NOT_SIMILAR" >

<Nuggets>

<LuaEventNugget

EventName="ExampleLuaEvent"

Radius="20.0"

SendToEnemies="true"

SendToAllies="true"

SendToNeutral="true">

</LuaEventNugget>

</Nuggets>

</WeaponTemplate>

3.通过 DrawModuleData 里的 LuaEvent：在动画进行时被触发。这个 Lua event 将会是"OnGenericEvent"并且 Lua 函数的第二个自变量值会的到一个 string 类型值，这个值等于 Data 里写的东西。下面是"ExampleAnimActive"的例子。更多的例子可以看 BFMEII 的 ini 源码。例子：

<Draws>

<ScriptedModelDraw id="ModuleTag_Draw">

...

<AnimationState

...

<Script>

AnyLuaCodeHere=true

</Script>

<LuaEvent Frame="0"

Data="ExampleAnimActive"

OnStateEnter="true"

OnStateLeave="false">

</LuaEvent>

</AnimationState>

</ScriptedModelDraw>

</Draws>

注：其实还有一种触发 OnGenericEvent 的方法，就是在切换技能的 <ToggleStatusSpecialAbilityUpdate>下属的 <ToggleState>中写 GenericLuaEventParameter=“*”，*就会成为 OnGenericEvent 的第二个参数，进而传递到 Lua 主函数的第二个参数。

4.Meta Mod 法：

用"run lua script"按钮手动运行 Lua 代码。

在 meta command 菜单的子菜单"write lua command"的帮助下运行 Lua 代码。

用代码计时器函数

把你的代码写进 PeriodicMasterChecker 函数。

把你的代码写进 AUTOSTART_CONFIG.lua 文件。

除了这些方法之外还有两样东西和 Lua 代码问题有关：

ForceLuaRegistration：设置 ForceLuaRegistration="True"，能把没有 AILuaEventsList 代码的 Object 写进 Lua 的 globals table。

DelayedLuaEventUpdate 的 behavior，用在《魔戒：中土之战 II》中

<DelayedLuaEventUpdate id="ModuleTag_LuaEventUpdate" />

我希望你能喜欢我的 mod，如果你是一个 modder 我希望你能根据教程用代码补完这个 mod。如果需要进一步的讨论我建议到 CNCNZ 论坛 http://forums.cncnz.com/forum/59-tiberium-wars-kanes-wrath-maps-modding/

联系方式：metaidea7@gmail.com

祝你好运
Mjjstral aka. MetaIdea (Maximilian F.)