**游戏脚本的实现**

  创建一个角色扮演游戏项目时，会发现在源码中编写游戏相关信息十分困难（这样做也是非常鲁莽的）。最好的办法就是使用外部数据源（类似于程序的代码），称之为游戏脚本（例如对话）。以这种方式，可以控制游戏的流程并节省宝贵的时间，因为不需要在每次做出改变后重新进行编译。

**理解脚本**

    当创建游戏时，游戏脚本与所编写的程序代码非常类似，只是游戏脚本相对于游戏引擎而言是外部的。正因为它们是外部的，所以才可以迅速地对脚本做出更改，而不用重新编译整个游戏引擎。否则对于一个超过100万行代码的项目，仅仅为了改变一个对话行就要重新编译整个项目。

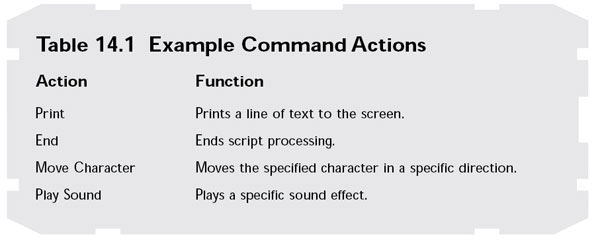
    脚本的使用并不会非常困难，而且游戏的每个方面都可以从脚本的运用中获益，比如导航菜单、战斗控制、处理玩家的物品清单，都可以使用脚本。举个例子，进行游戏开发时，把自己想象成处于战斗的用户，他们有规律地使用一系列的法术发起攻击。但在游戏开发过程中可能决定改变部分法术，如果法术的资料是硬编码的话，将面临一个非常麻烦的问题，必须更改控制法术的那些程序代码的每个实例，更不用说去调试和检验那些代码直到正确为止。为什么要花费如此多的时间去做这些改变呢？

    相反，可以将法术以及它们对游戏中人物的影响编写到几个小小的脚本里。每当战斗打响时，这些脚本被加载，并显示可供选择的法术。一旦该法术被施展，一个脚本将发挥自己的作用，从造成损伤到产生运动或法术图形的动画。

    有两种类型的脚本系统可供使用，第一种涉及到使用某种脚本编程语言，在脚本文件中输入命令，编译该文件，并在游戏中执行编译好的脚本文件。第二种是第一种的简化版本，将命令输入到一个文件中，系统通过从一个预先定义好的命令集里选择命令来创建脚本。

    为了简化问题的处理，我们使用第二种脚本系统来创建自己的脚本命令集，称之为Mad Lib Scripting（MLS）系统，它使用一个预先定义好的命令集合（称为行为action），同时每个定义好的命令都与一个游戏功能相关联。

    下图是一个脚本命令集示例：



    使用这样一个有限的行为集合，就不再需要复杂的可编译脚本语言了。相反，只需要告诉脚本系统要使用哪些行为，以及这些行为将使用怎样的选项以实现游戏的功能。对于这种方法，最大的好处就是不再需要为指定一个简单的行为而罗列代码行，可以通过编号来引用行为和选项。

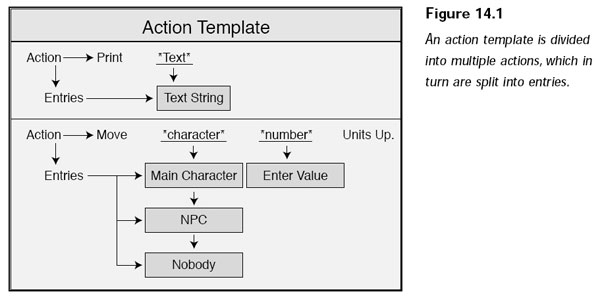
    举个例子，Play Sound行为的编号为4，而且该行为仅要求一个输入，即播放声音的编号。在脚本中只存储两个数值：一个对应于行为，另一个代表了声音。使用数值表示行为（代替文本）的方法可以使这种类型脚本的处理既快速又简单。

**Mad Lib Scripting系统的设计**

    创建在游戏中想到的行为，可以通过创建或编辑脚本来填充那些空白点（称之为条目entries）。对于每个行为，请明确提供一个可供空白条目填充的选项列表，它的类型可以从一行文本到一串数字。接着将行为和空白条目进行编号，以便脚本系统可以引用它们，以下是一些行为列表的范例：

1. Character (\*NAME\*) takes (\*NUMBER\*) damage.  
2. Print (\*TEXT\*).  
3. Play sound effect titled (\*SOUND\_NAME\*).  
4. Play music titled (\*MUSIC\_NAME\*).  
5. Create object (\*OBJECT\_NAME\*) at coordinates (\*XPOS\*),(\*YPOS\*).  
6. End script processing.

    在这6种行为中，都有0个或多个空白条目位于括号内，每个空白条目包含了一个文本字符串或者一个数字，这个行为与可能条目（以及条目的类型）的列表被称之为行为模板（action template），如下图所示：



    一旦使用了行为模板，就可以使用它们的编号而不是行为的文本进行引用（文本的存在只是为了使用户能够更容易理解每个行为所实现的功能）。

|  |  |  |
| --- | --- | --- |
| [http://i6.17173.itc.cn/job/images/index_2.gif](http://job.17173.com/index.php) | |  | | --- | |  | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 窗体顶端  http://i9.17173.itc.cn/job/images/index_5.gif | |  |  |  |  | | --- | --- | --- | --- | | [http://i1.17173.itc.cn/job/images/index_6.gif](http://job.17173.com/mem/fadv_list.php) | [http://i2.17173.itc.cn/job/images/index_7.gif](http://job.17173.com/mem/fresume_list.php) | [http://i3.17173.itc.cn/job/images/index_8.gif](http://job.17173.com/mem/showWork.php) | [http://i8.17173.itc.cn/job/2007/11/13/index_91.jpg](http://job.17173.com/zt/qzms.htm) | | **热门职位**：[**程序员**](http://job.17173.com/mem/sJobList.php?jobKey=%B3%CC%D0%F2%D4%B1) [**美工**](http://job.17173.com/mem/sJobList.php?jobKey=%C3%C0%B9%A4) [**市场**](http://job.17173.com/mem/sJobList.php?jobKey=%CA%D0%B3%A1) [**策划**](http://job.17173.com/mem/sJobList.php?jobKey=%B2%DF%BB%AE) | |  | | --- | | **[http://i1.17173.itc.cn/job/images/001.gif](http://job.17173.com/mem/reg.php?regkind=1)[http://i2.17173.itc.cn/job/images/002.gif](http://job.17173.com/mem/reg.php?regkind=2)** | | http://i1.17173.itc.cn/job/images/index_12.gif  窗体底端 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | **游戏脚本的实现** | | 2009年01月22日 14:19:38 　 【[发表评论/查看评论](http://comment1.news.17173.com/content.do?commentid=1752938&channel=90010&tn=90010200901&action=listtp" \t "_blank)】 |  |  | | --- | | **MLS系统的编写**      为了使MLS系统功能尽可能强大，需要设计它以便可以支持多重行为的模板，而且每个行为模板都包含不受数量限制的行为。以这种方式，就可以将系统复用到任何想要的项目中。当一个脚本完成时，将脚本读入到引擎中，并处理各自的行为，为每个由脚本编辑器所输入的行为使用指定的条目。      一个行为模板需要保存行为的列表，包括文本、条目编号以及每个条目的数据。每个行为按它们在列表中的索引值进行编号，同时每个行为中的空白条目也被加以编号。可以为每个条目指定一种类型（文本型、整数型、浮点型、布尔型、多重选择型），如下所示：  0. No entry type 1. Text entry 2. Boolean value 3. Integer number 4. Float number 5. Multiple choice (a choice from a list of text selections)      每个条目类型都有一个独特的特征，字符串类型的长度是可以变化的，数字型可以是两个数字范围之间的任何数值，而布尔值可以是TRUE或者FALSE。至于多重选项型，每个选项都有它自己的文本字符串（脚本从一个列表中获取选项，而且所选选项的索引编号比它的文本更适用）。      行为可以采用如下格式:  Action #1: Spell targets (\*MULTIPLE\_CHOICE\*).  Possible choices for blank entry #1: 1. Player character 2. Spell caster 3. Spell target 4. Nobody      我们通过创建结构体ENTRY\_RULE和ACTION来处理条目规则与行为。  enum ENTRY\_TYPE { ENTRY\_NONE = 0, ENTRY\_TEXT, ENTRY\_BOOL, ENTRY\_INT, ENTRY\_FLOAT, ENTRY\_CHOICE };  typedef char\* char\_ptr; typedef int   BOOL;  //============================================================================ // Structures to store information about a single blank entry. //============================================================================ typedef struct ENTRY\_RULE {     long    type;     // type of blank entry (ENTRY\_TEXT, ENTRY\_BOOL, )      // The following two unions contain the various information about a single blank entry,      // from the min/max values (for int and float types), as well as the number of choices      // in a multiple choice entry.     union     {         long        long\_min;       // min value of long type         float       float\_min;      // min value of float type         long        num\_choices;    // number of choices in list     };      union     {         long        long\_max;       // max value of long type         float       float\_max;      // max value of float type         char\_ptr\*   choices;        // choice text array         };      // structure constructor to clear to default values     ENTRY\_RULE()     {         memset(this, 0, sizeof(\*this));     }      // structure destructor to clean up used resources     ~ENTRY\_RULE()     {         // special case for choice type         if(type == ENTRY\_CHOICE && choices != NULL)         {             for(long i = 0; i < num\_choices; i++)                             delete[] choices[i];                              delete[] choices;                     }     } } \*ENTRY\_RULE\_PTR;  //============================================================================ // Structure that store a single action. //============================================================================ typedef struct ACTION {     long            index;              // action index [0, number of action - 1]     char            text[256];          // action text      short           num\_entries\_rule;   // number of entries in action     ENTRY\_RULE\_PTR  entries\_rule;       // array of entry structures      ACTION\*         next;               // next action in linked list      ACTION()     {         memset(this, 0, sizeof(\*this));     }      ~ACTION()     {         delete[] entries\_rule;         delete next;     } } \*ACTION\_PTR; | | |

行为模板被存储为文本文件，同时每个行为的文本被包括在括号中。每个包含条目的行为（标记为文本中的波浪字符）紧跟着是条目数据的列表。每个条目由一个描述条目类型（文本型、布尔型、整型、浮点型或选项型）的单词开始。对于文本类型而言并没有更多的需要信息，对于布尔类型来说也是如此。而作为整数和浮点型，则要求一个最小值和最大值。最后，选项类型条目后跟着的是可供选择的编号以及每个选项的文本（包括在引号里）。

    如下所示：

"If flag #~ is ~ then"  
INT 0 255  
BOOL

"Else"

"Endif"

"Set flag #~ to ~"  
INT 0 255  
BOOL

"Print ~"  
TEXT

"Move character to ~, ~, ~"  
FLOAT 0.0 2048.0  
FLOAT 0.0 2048.0  
FLOAT 0.0 2048.0

"Character ~ ~ ~ ~ points"  
CHOICE 3   
"Main Character"   
"Caster"   
"Target"  
CHOICE 2  
"Gains"  
"Looses"  
INT 0 128  
CHOICE 2  
"Hit"  
"Magic"

"Engage in battle sequence #~"  
INT 0 65535

"End Script"