騎馬與砍殺

MOD系统教材—中文版

骑马与砍杀中文站论坛：**http://bbs.mountblade.com.cn/**

汉化名单

此教程中的内容相当部分沿用**0.808** 版教程，所以此次汉化的相当部分也参考了**0.808** 版教程的汉化。

0.808 版教程汉化名单：

abinchen（第1,2,3,4,5,7,9 部分）

waterwande （第6,8 部分）

以下为此次汉化名单：

asdolagun（第7 部分）

Ginn（第11,12,附录部分）

James（第10 部分）

foxyman（第8 部分）

Hetairoi（第1~6,9 部分）

校对：

Foxyman

Hetairoi

官方原文发布地址：

http://forums.taleworlds.net/index.php/topic,56798.0.html

目录

第一部分

1.1 Mod 系统是什么?

骑马与砍杀Mod 系统是一组Python 脚本，通过这些脚本可以生成新的游戏内容和修改已有的内容。我们的官方原版Mod 就是使用这个系统来创建的。你可以利用它来增加新兵种、新人物、新任务、新对话等，并可以修改现有的游戏元素。

需要注意的一点的是，M&B 并不直接读取Mod 的Python 脚本，而是读取由Python 脚本执行生成的文本文件。

骑马与砍杀实际上从Mount&Blade/Module 文件夹下的文本文件中读取内容。因此，理论上你可以通过修改这些文本文件来完成所有的Mod 内容修改（实际上，一些Mod 作者已经知道了如何利用这些文本文件并开发出了绝妙的Mod）。但是这些文本文件可读性相当差，直接修改它们并不实际。开发新的Mod 目前有两种方法，一种是使用本文提到的官方Mod 系统。另一种是Effidian 的非官方编辑器，不过现在已经废止并且不能用于最新的1.011 版，只支持一些老的版本，比如0.751 版。

1.2 使用Mod 系统的必要条件:

Mod 系统由Python 脚本构成，所以你需要安装和配置好Python。以下地址可以下载到Python。

http://www.python.org/download/

该页面上有若干个下载链接，你只需下载最新的Windows 安装版2.6.1 即可。

注意：其它版本的Python 都不行。必须得是2.6 版。（Hetairoi 注：其实不一定非2.6 版不可，我一直用的2.5 版挺好的，不过3.0 及以上版本的确不行）

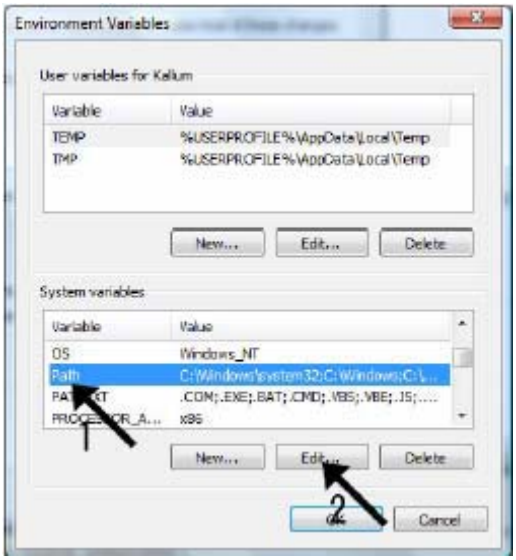
下载并安装完毕Python 后，需要在Windows 环境变量中设置一下。这点非常重要，请确保正确完成此设置。

对于Windows 9x，编辑 autoexec.bat 文件，把Python 文件夹添加进Path 中。例如，Python

安装在C:\Python24，则添加如下语句：

```
set PATH=C:\Python24;%PATH%
```

如果是Windows XP， 操作稍有不同: 右键点击“我的电脑”，选择“属性”，点击“高级”页，点击“环境变量”按钮：



1.在“系统变量”框的右边按下拉条，找到‘Path’变量。

2.选中‘Path’按下面的“编辑”按钮，弹出新的对话框：

移动光标到“变量值”框的末端然后输入“;C:\Python26”，这里的“;C:\Python26”是你的Python 安装目录。（Hetairoi 注：原文如此，老外一会儿Python24 一会儿Python26 的，看来是拷贝粘贴的时候没注意）然后两次点击“确认”完成。

1.3 获取Mod系统

最新版的Mod 系统可从官方网页下载：

http://www.taleworlds.com/download/mb_module_system_1010_0.zip

下载zip 文件并且解压缩（可能需要WinRAR，7-zip 之类的软件）。把MOD 系统文件解压到容易找得到的地方，比如，桌面或者我的文档。

1.4 Mod系统的文件

现在，我们来看一下Mod 系统包含的文件。观察所有的Python 文件 (由.py 结尾的文件)，我们可以看到四种文件:

* header_ 前缀的文件

* process_ 前缀的文件

* ID_前缀的文件

* module_前缀的文件

前两类文件是运行Mod 系统所必须的，不要修改它们。 第三类文件(ID_)是生成Mod 时建立的临时文件，可以删除之，系统之后会自动生成这些文件。最后一类文件 (module_)是实际包含Mod 数据内容的，你将要编辑的就是这些文件。

1.5 建立新Mod

首先我们来为新Mod 建立一个文件夹。进入Mount&Blade/Modules 文件夹(默认位于"C:/ProgramFiles/Mount&Blade/Modules")。现在，Modules 文件夹下应当有一个名为Native的文件夹，这就是所谓的官方原版Mod。你需要给你的新mod 在这个位置创建一个新文件夹，然后复制Native 下所有的文件到新文件夹中。此文件夹将是你自己的Mod 文件夹，所以你可以任意给它一个名字。简单起见，我这里假设它的名字为MyNewModule。

你可以启动骑马与砍杀来看你做得正确与否。现在，M&B 的启动窗口应该有一个下拉框让你选择Mod。

试着选择你的新Mod 然后启动一个新游戏。由于我们是从native 文件夹下拷贝了所有内容，我们进入的游戏应该和原版完全一样。

下面，我们需要把Mod 系统的目标文件夹设成刚才创建的新文件夹。编辑module_info.py (右键点击此文件，选择"Edit with IDLE";或者用记事本以及你爱用的文本编辑器亦可) ，更改export_dir 指向新文件夹。比如，如果你的Mod 文件夹为: c:/Program files/Mount&Blade/Modules/MyNewModule ，则修改这一行为:

```
export_dir = "C:/Program Files/Mount&Blade/Modules/MyNewModule/"
```

注意：在MOD 系统里面，目录名是用正斜杠(/)而不是用反斜杠(\)隔开的。另外注意在那句指向目录语句的末尾也需要一个正斜杠。

现在我们的Mod 系统已经配置好了。让我们试着删除新文件夹中的conversation.txt，然后双击build_module.bat。你应当会看见一个命令提示符和类似下面的输出：

```
Initializing...
Compiling all global variables...
Exporting strings...
Exporting skills...
Exporting tracks...
Exporting animations...
Exporting meshes...
Exporting sounds...
Exporting skins...
Exporting map icons...
Creating new tag_uses.txt file...
Creating new quick_strings.txt file...
Exporting faction data...
Exporting item data...
Exporting scene data...
Exporting troops data
Exporting particle data...
Exporting scene props...
Exporting tableau materials data...
Exporting presentations...
Exporting party_template data...
Exporting parties
Exporting quest data...
Exporting scripts...
Exporting mission_template data...
Exporting game menus data...
exporting simple triggers...
exporting triggers...
exporting dialogs...
Checking global variable usages...
------------------------------
Script processing has ended.
Press any key to exit. . .
```

如果你碰到了错误，请确保你严格按照本教程的步骤做了。如果你确实这样做了，请使用官方论坛的搜索功能，很有可能已经有人遇到了同样的问题并发布了简单的解决方法。

如果一切都顺利，现在检查你的新Mod 文件夹，其中应当有一个新的conversations.txt 文件。恭喜你！你已经成功设置了M&B Mod 系统！请接着看教程第二部分。

第二部分

上一章中提到，Mod 系统是这样使用的：

1）编辑一个或多个Mod 文件（以module 开头，.py 结尾的文件），然后按照你的喜好来修改它们。 (通常你需要右键点击并选择'Edit with Idle'来进行修改)

2）接着，双击build_module.bat。这个操作将尝试建立你的Mod（并且报告任何存在的错误）

3）若没有任何错误，你可以启动M&B 测试你更改的内容。有时候你需要创建一个新游戏来使更改生效。

注意：尽管PYTHON 语言有它自己的编辑器，我发现使用NOTEPAD++更有效。你可以在不同的标签下打开多个文件，还可以定制界面。通过和其它有着不同版面代码编辑器（比如BYOND）相对比我发现这个很好用。

2.1 个性化你的MOD

开始之前，我们首先个性化你的MOD。然我们更改MOD 选择界面显示的图片。这个可以用WINDOWS画图软件或者其它任何能修改BMP 文件的软件。这个图片文件（main.bmp）位于你骑马与砍杀剧本文件夹内（比如：C:\Program Files\Mount&Blade\Modules\MyMod）。修改这个会改变MOD 选择界面下选择你的MOD 时显示的图片。这个对于当你制作一个MOD 的同时又在玩另一个MOD 时是个好做法。

如果你会修改DDS 图像文件，你可以拷贝多个DDS 背景图片到你的MOD 的纹理文件目录（比如，从：C:\Program Files\Mount&Blade\Textures 到 C:\Program Files\Mount&Blade\Modules\MyMod\Textures）。如果你修改了这些文件，当你玩你的MOD 的时候会看到相应的改变。不要在这方面太花精力，只要知道当做了这些会让你的MOD 看上去非常花哨。

2.2 编辑Mod 系统文件

Mod 系统使用Python 列表来表示游戏对象的集合。(一个Python 列表以 [符号打头，包括一列由逗号分隔的对象，并且以] 符号结尾) ，你打开任意一个mod 文件，都会发现它包含这样一个列表。例如 module_map_icons.py 中有：

```
map_icons = [
("player",0,"player", avatar_scale, snd_footstep_grass, 0.15, 0.173, 0),
("player_horseman",0,"player_horseman", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("gray_knight",0,"knight_a", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("vaegir_knight",0,"knight_b", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("flagbearer_a",0,"flagbearer_a", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("flagbearer_b",0,"flagbearer_b", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("peasant",0,"peasant_a", avatar_scale,snd_footstep_grass, 0.15, 0.173, 0),
```

("khergit",0,"khergit_horseman", avatar_scale,snd_gallop, 0.15, 0.173, 0),
("khergit_horseman_b",0,"khergit_horseman_b", avatar_scale,snd_gallop, 0.15, 0.173, 0),
("axeman",0,"bandit_a", avatar_scale,snd_footstep_grass, 0.15, 0.173, 0),
("woman",0,"woman_a", avatar_scale,snd_footstep_grass, 0.15, 0.173, 0),
("woman_b",0,"woman_b", avatar_scale,snd_footstep_grass, 0.15, 0.173, 0),
("town",mcn_no_shadow,"map_town_a", 0.35,0),
("town_steppe",mcn_no_shadow,"map_town_steppe_a", 0.35,0),
("village_a",mcn_no_shadow,"map_village_a", 0.45, 0),
("village_burnt_a",mcn_no_shadow,"map_village_burnt_a", 0.45, 0),
("village_deserted_a",mcn_no_shadow,"map_village_deserted_a", 0.45, 0),
###还有很多，只显示这些了###

这里map_icons 被声明为一个Python 列表(list)，列表中的每一个元素都是一个特定的地图图标对象的声明。在这个例子中，("player",0,"player", avatar_scale, snd_footstep_grass, 0.15, 0.173, 0), 就是这样一个对象。

我们称其为元组(tuple)。元组和列表类似，包括由逗号分隔的元素（但是它们以括号作为开头和结尾）。每个元组对象的结构在mod 文件的开头定义。对于地图图标，每个元组对象包含：

1）图标名，每个图标id 的前面会自动加上前缀icon_
2）图标标识，查看header_map icons.py 文件列出的可用标示
3）网格(mesh)名，可以在BRF 文件map_icon_meshes，map_icons_b，和map_icons_c 中找到
4）网格缩放(scale)，
5）声音id，
6）图标标识的x 方向偏移量
7）图标标识的y 方向偏移量
8）图标标识的z 方向偏移量

因此，对第一个元组来说：

("player",0,"player", avatar_scale, snd_footstep_grass, 0.15, 0.173, 0),

1）图标名= "player"
2）图标标识= 0
3）网格名= "player"
4）网格缩放= 0.2
5）声音id = snd_footstep_grass
6) x 方向偏移量 = 0.15
7) y 方向偏移量 = 0.173
6) z 方向偏移量 = 0

通过使列表的内容与文档开头部分的定义相匹配，你可以创建任意mod 系统文件中的游戏对象结构。

2.3 添加新的游戏对象

了解清楚地图图标元组的结构之后，就可以添加我们自己的地图图标了。让我们再看一遍这个列表。

```
map_icons = [
("player",0,"player", avatar_scale, snd_footstep_grass, 0.15, 0.173, 0),
("player_horseman",0,"player_horseman", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("gray_knight",0,"knight_a", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("vaegir_knight",0,"knight_b", avatar_scale, snd_gallop, 0.15, 0.173, 0),
("flagbearer_a",0,"flagbearer_a", avatar_scale, snd_gallop, 0.15, 0.173, 0),
.
.
.
("banner_125",0,"map_flag_f20", banner_scale,0),
("banner_126",0,"map_flag_15", banner_scale,0),
]
###还有很多，只显示这些了###
```

任何一个mod 文件中的新游戏对象必须添加在这个列表内。你可以看到module_map_icons 列表刚好在("banner_126",0,"map_flag_15", banner_scale,0)) 后结束。为了给我们的新游戏对象腾出空间，必须把后括号往下移一行。

做好上一步后，就可以添加新的对象了。最简便的办法是复制和粘贴一个已存在的对象，然后编辑其内容。

例如，在”banner_126”后面拷贝”town”元组：

```
("banner_126",0,"map_flag_15", banner_scale,0),
("town",mcn_no_shadow,"map_town_a", 0.35,0),
]
```

这个例子中，我们拷贝了("town",mcn_no_shadow,"map_town_a", 0.35,0) 并给了它一个新名字"new_icon"。

这个新图标具有一个标识。标识是一些可以在相应的地方打开或关闭的设置，可以通过包含或去掉这些标识来打开或关闭它们。例如，mcn_no_shadow 这个标识使用在我们的新图标上，会使图标不投下阴影。

我们现在从新图标上删去mcn_no_shadow 标识，把它替换为0，这样就告诉mod 系统这个图标没有任何标识。

```
("banner_126",0,"map_flag_15", banner_scale,0),
("new_icon",0,"map_town_a", 0.35, 0),
]
```

"town"和"new_icon"都使用了"map_town_a"网格，这意味着它们都使用了游戏资源文件中名为map_town_a

的3D 模型。更改这个字段可以让新图标使用资源文件中的任意3D 模型。因为两个图标都使用了同一个网格，如果我们现在把"new_icon"放进游戏，它将和"town"看起来是一个模子里出来的。

现在我们来稍微改变一下"new_icon"的外观。我们把它改成"city"。这个网格没有起用，它将在我们的MOD里面看上去更加突出。

```
("banner_126",0,"map_flag_15", banner_scale,0),
("new_icon",0,"city", 2,0),
]
```

这个例子中，我们把图标的缩放从0.35 更改为2。这意味着此图标将会以通常大小的两倍来显示。在游戏中，这可以帮助我们把它和"town"区分开。

接下来我们在module_parties.py 里面创建一个使用我们的新图标的队伍(party)。为此我们需要从module_parties.py 中引用这个图标。

在继续之前，我们需要运行一下build_module.bat。这样可以让我们知道我们在之前的编辑中是不是有语法错误。不时运行一下这个build 是个检查错误的好办法，否则越拖到后面检错就会变得越困难。

2.4 引用游戏对象

在你的mod 系统文件夹中打开module_parties.py 文件，将会看到另一个Python 列表，parties = [，就在一些常量声明的后面(pf_town = pf_is_static | pf_always_visible | pf_show_faction | pf_label_large)。常量声明可以使得输入大量标示变得很方便。关于常量后面还会说到。

正如你看到的，module_parties.py 中的元组和module_icons 稍有不同。许多元组（如果不是全部的话）都是这样。我们利用这个机会仔细看看队伍的结构。首先，了解队伍（party）可以是任何你在地图上可以碰到的东西。可以是移动的部队或者是固定的村庄、城镇或者其它你自己定义的场景。关于队伍（party）的一个关键点是当两个或以上的party 相遇的时候会触发相应的事件。

让我们看看队伍的一个例子。如果你向下多拉一点你会看到：

```
("town_1","Sargoth", icon_town|pf_town, no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(-1.55, 66.45),[], 170),
```

这个元组把萨哥斯（Sargoth）放置在地图上。萨哥斯的多项属性在合适的字段中进行配置——非常类似我们在module_icons.py 中看到的字段。

元组字段解析：

1）队伍id。队伍在其他文件中被引用时的id。
2）队伍名。队伍在游戏中表现的名字。可以根据你的喜好设成跟队伍id 不同的名字。
3）队伍标识(flag)。每个队伍对象的第一个标识必须为此队伍所用的图标。
4）菜单。这个字段已经被摈弃了，它对M&B 0.730 以上版本已经不再有效。
5）队伍模板。此队伍从属的队伍模板id。pt_none 为其默认值。
6）队伍阵营(faction)。这可以是module_factions.py 中的任意条目。

7) 队伍个性。个性标识的解释详见header_parties.py。
8) AI 行为。队伍在大地图上的AI 行为表现。
9) AI 目标队伍。AI 行为的目标。
10) 初始坐标。队伍在大地图上的初始坐标X, Y。
11) 部队栏(troop stack)列表。每个部队栏是一个三元组，包含以下字段：
11.1) 部队id。可以是module_troops.py 中的常规兵种或英雄。
11.2) 此部队栏中的数量，是固定的。你在这里输入的数字就是城镇将有的部队数量。
11.3) 成员标识，可选的。可以使用pmf_is_prisoner 来标注俘虏。
12) 队伍方向的度数，可选。

萨哥斯元组的解释：

("town_1","Sargoth", icon_town|pf_town, no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(-1.55, 66.45),[], 170),

1) 队伍id = "town_1"
2) 队伍名= "Sargoth"
3) 队伍标识= icon_town|pf_town
4) 菜单= no_menu
5) 队伍模板= pt_none
6) 队伍阵营= fac_neutral
7) 队伍个性= 0
8) AI 行为= ai_bhvr_hold
9) AI 目标队伍= 0
10) 初始坐标= (-1.55, 66.45)
11) 部队栏列表= [] (None)
12) 队伍方向 = 170

观察第3 个字段，我们可以看到萨哥斯通过加入icon_前缀，引用了module_icons.py 中的“town”。Mod系统通过这个前缀指向正确的mod 文件。要引用module_icons，我们用icon_;要引用module_factions，我们用fac_;要引用module_parties，我们用p_；诸如此类。每个mod文件都有和其对应的前缀——这一章的结尾列出了所有前缀。

既然我们已经知道了队伍的结构，就可以来添加自己的队伍了。但是开始之前，请注意：module_parties.py和其他一些特定的mod 文件中，你不可以在列表的尾部添加新的城镇。这些文件中有相关注释警告你不要这样做，因为这会打断native 代码的执行。在module_parties.py 中建议你在“town_1”和“castle_1” 之间添加新的条目。这在module_constants.py 里面有详细说明，稍后解释。

现在，复制条目“town_1” 然后把它粘贴到“town_18”后面，“castle_1”的前面。

("town_18","Narra", icon_town_steppe|pf_town, no_menu, pt_none, fac_neutral, 0, ai_bhvr_hold, 0, (-22.6, -82), [], 135),
##JIK's Test Area
("town_1","Sargoth", icon_town|pf_town, no_menu, pt_none, fac_neutral, 0, ai_bhvr_hold, 0, (-1.55, 66.45),[], 170),
##End of JIK's Test Area
Aztaq_Castle

```
# Malabadi_Castle
("castle_1","Culmarr_Castle",icon_castle_a|pf_castle, no_menu, pt_none, ac_neutral, 0,
  ai_bhvr_hold, 0, (-69.2, 31.3), [], 50),.
.
```

这里我简要说明一下关于文档和注释内容。就像你看到的我之前注释了我将要使用的部分并且在前面用加上#和我的名字做了标记。这有助你找到你正在修改的地方（使用大多数编辑器拥有的“查找”功能）。用注释来说明你正在修改什么内容，或者如果显示不出你正在修改的地方那就说明会牵涉到什么也是个很好的主意。任何在#符号后面的部分编译器都会忽略掉，这样你可以注释掉整行，或者在代码行的末尾增加注释。这可以有助你编辑你的mod，也同样可以帮助那些打算学习你的作品的人。在这份指南里，我会开始加入一些简单的注释。让这成为一种习惯。

这个例子中，我们把队伍的id 从”town_1”改为”mod_town”，队伍名从”Sargoth”（萨哥斯）改为”Mod_Town”。

通过观察这个元组，我们现在可以确定：

1）从另一个文件引用这个队伍，我们必须使用id”mod_town”和前缀”p_”，即”p_mod_town”。
2）在游戏中，我们只会见到”Mod Town”这个描述队伍的名字，而非它的id。
3）这个队伍使用icon_town 图标和pf_town 标识——此标识定义了通用的城镇设置。我们下一步将修改这个标识。
4）Mod Town 当前属于neutral（中立）阵营。
5）如果我们现在把这个新城镇放进游戏，它将和Sargoth 出现在地图的同一个坐标。这个我们接下来也要更改。

```
##JIK's Test Area
("mod_town","Mod_Town", icon_new_icon|pf_town, no_menu, pt_none, ac_neutral, 0,
ai_bhvr_hold, 0, (-1, -1), [], 45),
##End of JIK's Test Area
```

这里我们将新城镇的地图坐标改为(-1,-1)，方向改成45，让它看上去有些与众不同。这个城镇现在使用了我们的新图标，”new_icon”，拥有自己独一无二的地图坐标，这使得它可以毫无问题地显示出来。

保存，然后执行build_module.bat。如果一切顺利，启动你的mod 看看地图中间附近出现的新城镇和图标。

试试吧。

如果不是一切顺利，仔细检查一下拼写和语法。确保所有的逗号和括号都在恰当的位置。语法错误是原版mod 中编译器发生错误的最常见根源。编译时，错误的地方通常会用一种尖角符号标志出来。使用一种代码编辑器比如NOTEPAD++可以在行比较有限的情况下让编写比较容易，它可以高亮显示（当鼠标悬停某处）括号起始和结束部分。

游戏中，到新城镇时会触发城镇菜单。你可以试着访问酒馆，或者到街上逛逛，但是你会发现场景时错误的。我们已经把新的队伍放置在了地图上，但是我们还没有完善它的内部。稍

后的章节中我们会回到这里，那时我们会为一个小剧情建立一个酒馆。我们先把它放着，一步一步地来……

正如你所看到的，不同的mod 文件之间的交互是十分广泛的。你的mod 要想正常工作必须覆盖所有的环节。幸运的是，多数更改最多只需编辑一到两个文件。

既然你已经掌握了制作mod 的基础，我们现在可以更进一步地学习不同的mod 文件了。知道如何引用别的MOD 文件里面的项目非常重要，你需要知道不同项目的前缀。以下列出了一部分。各个项目文件类型前缀可以在各个**module_<*type*>.py** MOD 文件开头的元组说明信息中看到。

mod 文件前缀列表：

fac_ -- module_factions.py
icon_ -- module_map_icons.py
itm_ -- module_items.py
mnu_ -- module_game_menus.py
mno_ -- module_game_menus.py——在module_game_menus 中引用单独的菜单项，现在可能已经无效
mt_ -- module_mission_templates.py
psys_ -- module_particle_systems.py
p_ -- module_parties.py
pt_ -- module_party_templates.py
qst_ -- module_quests.py
script_ -- module_scripts.py
scn_ -- module_scenes.py
spr_ -- module_scene_props.py
str_ -- module_strings.py
trp_ -- module_troops.py
skl_ -- module_skills.py
module_dialogs.py 从不直接被引用，所以它没有前缀。

第三部分

这一章我们讨论module_troops.py 及其功能。Module_troops 中定义所有的常规兵种，英雄，箱子和城镇NPC，包括他们的容貌，能力数值和装备库。要想创建一个新人物或兵种，则修改这个文件。

3.1 Module_Troops 解析

这个文件开头的代码块是用来计算武器熟练度的，以及其他一些不能mod 的代码。这块代码在Python 列表区域之外，所以我们无须编辑它也不用关心它。直接跳到列表troops = [吧。

这里我们找到一些元组定义了游戏中的玩家和几个游戏中非常重要的兵种。紧跟着的是我们在训练场遇到的多个战士。我们来学习一下这些东东，因为这是常规兵种成长升级的很好范例。

观察以下内容：

["novice_fighter","Novice Fighter","Novice Fighters",
tf_guarantee_boots|tf_guarantee_armor,no_scene,reserved,fac_commoners,
[itm_hide_boots],
str_6|agi_6|level(5),wp(60),knows_common,mercenary_face_1, mercenary_face_2],

这是一个垃圾兵种“novice fighter”（初级斗士）。“novice fighter”是低等兵种，不擅长战斗，拥有低能力值，名不见经传。

元组字段解析：

1）兵种id。用来在其他文件中被引用。
2）兵种名。
3）兵种名复数形式。
4）兵种标识。如果你想保证这个兵种永远有一类武器装备的话，必须设置tf_guarantee_*标识。否则，此兵种出现的时候可能没有该类装备，如果该兵种的物品库中有近战武器，则只有近战武器会保证被装备上。
5）场景(scene) 。这只对英雄适用， 定义了英雄将出现在哪个场景和哪个入口。举个例子，
scn_reyvadin_castle|entry(1) 把部队放入Reyvadin（日瓦丁）城堡的入口1。
6）预留。目前没用到，值为reserved 或 0。
7）阵营。兵种的阵营，以fac_为前缀。
8）装备库(inventory)。兵种装备库的一系列物品。常规兵种会随机从这里选择装备。
9）属性(attribute)。兵种的属性值和人物等级，和玩家属性的工作机制一样。
10）武器熟练度。该兵种的武器熟练度值。wp(x)函数将建立一个接近x 值的随机武器熟练度，当然你也可以针对特定的熟练度添加附加定义。如，要创造一个其他武器的熟练度为60 左右的资深弓箭手，你可以定义如下：

wp_archery(160) | wp(60)

（ # 注意： 这个算法有问题！ 比如原版里面的库吉特资深骑射手按照这种方法来定义的wp(90)|wp_archery(130)，结果其弓箭熟练度在200 以上。

这是**foxyman** 对此算法的解释：

“如果确实想要实现类似的效果：其他的熟练度在90 左右，弓箭在130 左右的话，可以这么写：

wp(80)+wp_archery(130-80)

如果还要定义另一个加在后面就可以了：

p(80)+wp_archery(130-80)+wp_one_handed(150-80)另外熟练度的上限是1023”

这里是**oolonglgx** 对此算法的进一步解释：

“在module_troops.py 开头关于这个地方的注释：

10) Weapon proficiencies (int): Example usage:

#

wp_one_handed(55)|wp_two_handed(90)|wp_polearm(36)|wp_archery(80)|wp_crossbow(24)|wp_throwing(45)

这样能够精确定义各种武器的熟练度。

再往下可以看到：

```
def wp_melee(x):
n = 0
r = 10 + int(x / 10)
n |= wp_one_handed(x + random.randrange(r))
n |= wp_two_handed(x + random.randrange(r))
n |= wp_polearm(x + random.randrange(r))
return n
```

所以一个近战武器熟练度60、弓箭熟练度150 的资深弓箭手还可以这样写：

wp_melee(60)|wp_archery(150)”）

11）技能(skill)。这和玩家技能一样。注意，在你定义的属性和技能之外，兵种的每一个人物级别也会得到一个随机属性点和一个随机技能点。

12）容貌代码(face code)。游戏会根据这个代码生成容貌。你可以在游戏中的容貌调节窗口按**CTRL+E** 来输出容貌代码，前提是打开编辑模式。

13）容貌代码2。只对常规兵种适用，对英雄可以忽略。游戏将为该兵种的每个个体生成容貌代码1 和容貌代码2 之间的随机容貌特征。

检查Novice_fighter 元组：

1）兵种id = "novice_fighter"
2）兵种名= "novice_fighter"
3）兵种名复数形式= "novice_fighters"
4）兵种标识= tf_guarantee_boots|tf_guarantee_armor
5）场景= no_scene
6）预留= reserved
7）阵营= fac_commoners
8）装备库= [itm_sword,itm_hide_boots]
9）属性= str_6|agi_6|level(5)
10）武器熟练度 = wp(60)

11）技能= knows_common
12）容貌代码= swadian_face1
13）容貌代码2 = swadian_face2

关于这个元组，有三样东西是没有作用的。

我们的"novice fighter"设置了tf_guarantee_armor（保证防具），但是他自己没有防具，然而这并不使tf_guarantee_armor 成为多余；部队会穿上他在游戏中获得的任意防具。

刚开始的时候（即第1 级）,"novice_fighter"力量为6，敏捷为6。游戏启动后，他一下子跳到第5 级，同时得到了隐式定义的所有点数（即5 个属性点和5 个技能点）。

他有技能knows_common（懂得普通技能）。knows_common 是module_troops 开头部分定义的技能集；现在向上翻滚文档并观察这个集合。

knows_common = knows_riding_1 | knows_trade_2 | knows_inventory_management_2 | knows_prisoner_management_1 | knows_leadership_1

赋予了knows_common 的兵种将拥有这里列出的所有技能：1 点骑术，2 点交易，2 点物品管理，1 点俘虏管理和1 点统御。knows_common 是一个常量——代表了其他一些东西的词：如数字，id，另一个常量，或另一个合法对象。一个常量可以代表任意数量的对象，只要那些对象在你需要它们出现的地方以正确的顺序出现。

这个例子中，knows_common 被定义为会等号右侧的所有技能。因此把knows_common 放进技能字段的效果是, mod 系统的运行方式将和你在技能字段把那些技能全部手工输入一遍一样。

现在，让我们看列表中的下一个条目。

["regular_fighter","Regular Fighter","Regular Fighters", tf_guarantee_boots | tf_guarantee_armor, no_scene, reserved, fac_commoners,
[itm_hide_boots],
str_8 | agi_8 | level(11), wp(90), knows_common | knows_ironflesh_1 | knows_power_strike_1 | knows_athletics_1 | knows_riding_1 | knows_shield_2, mercenary_face_1, mercenary_face_2],

这个例子中，你可以看到稍强一点的“regular fighter”（普通斗士），他拥有较高的能力值，11 级，掌握knows_common 之外的一些技能。游戏中，如果我们的团队中有“novice fighters”到达11 级所需的经验点，我们可将其升级为“regular fighters”。下面，我们将看看怎么来设置升级的。

3.2 升级部队

什么兵种可以升级成什么是在module_troops 的尾部定义的。相关的定义包括那些兵种可以升级，可以升级为那个兵种，以及相应的升级所需等级。

可以看到，每个兵种的升级方案须在这里用操作符upgrade(troops)定义。第一个字符串是将要被升级的兵种id，第二个字符串是升级后的兵种id。举个例子来说，upgrade(troops,"farmer", "watchman") 将允许一个"farmer"(农民)升级成 "watchman"（趟子手），当"farmer" 得到足

够经验点数的时候有两种升级操作。

upgrade(troops,"source_troop", "target_troop_1") 提供了唯一的升级选择； "source_troop" 到 "target_troop_1"。

upgrade2(troops,"source_troop", "target_troop_1", "target_troop_2") ， 则提供了把 "source_troop" 升级到"target_troop_1" 或者 "target_troop_2"的2 种选择。 2 种是当前可能的升级选择之最大数量。

目前在这个区块中没有"novice_fighter"的条目，所以我们来创造一个。复制 upgrade(troops,"farmer", "watchman")并粘贴到区块的底部。注意升级树部分在主要的兵种代码块以外（在最后一个] 符号的后面）。

注意代码所在位置很重要，因为编译器是按照某种特定的顺序来看代码的。同样要注意这一部分代码的每一个行后面没有逗号。要时刻注意你正在编写哪一部分的代码，和相应的编写规则。

然后改"farmer" 为 "novice_fighter" ，改 "watchman" 为 "regular_fighter"。我们的团队中任何 "novice fighters" 现在将可以升级为"regular fighters"， 正如最后一段代码中所描述的。

接着， 我们来做一点修改。在列表最后再添加一个条目， 源兵种为 "new_troop" ,目标兵种为"regular_fighter"。“new_troop” 现在还不存在，但很快就会被创建！向上滚动文档到：# Add Extra Quest NPCs below this point。新的兵种应该加在”local_merchant”条目之前，我们现在就来干这个。

3.3 添加新兵种

在# Add Extra Quest NPCs below this point 下面加几个回车，然后拷贝/粘贴”novice_fighter”的代码到空白区域并且作如下改变：

```
##JIK - new troop entry
["new_troop","new_troop","new_troops",tf_guarantee_boots|tf_guarantee_armor,no_scene,reserved,fac_commoners,
[itm_sword_medieval_a, itm_fighting_axe, itm_leather_jerkin, itm_skullcap, itm_hide_boots],
str_6 | agi_6 | level(5), wp(60), knows_common, mercenary_face_1, mercenary_face_2],
```

从现在开始，每个"new_troop" 种类的部队将穿着itm_leather_jerkin（皮短袖紧身衣）。他们还会随机地从itm_sword_medieval_a 和itm_fighting_axe 当中挑选一件来挥舞。然而，由于标识字段中定义的条目，他们中只有部分人有itm_skullcap（头顶盔）；这个兵种只确保了防具和靴子。为了保证我们的新兵种都戴着头盔，我们必须把tf_guarantee_helmet（保证头盔）加到标识字段中去。我们来照做。

```
##JIK - new troop entry
["new_troop","new_troop","new_troops", tf_guarantee_boots | tf_guarantee_armor |
tf_guarantee_helmet, no_scene, reserved, fac_commoners,
[itm_sword_medieval_a, itm_fighting_axe, itm_leather_jerkin, itm_skullcap, itm_hide_boots],
str_6 | agi_6 | level(5), wp(60), knows_common, mercenary_face_1, mercenary_face_2],
```

接下来我们修改兵种的属性。把STR（力量）设为9，AGI（敏捷）设为9。这些做好后，改人物等级为4级，武器熟练度为80。

我们的"new troop"看起来应当像这样：

```
##JIK - new troop entry
["new_troop","new_troop","new_troops", tf_guarantee_boots | tf_guarantee_armor |
tf_guarantee_helmet, no_scene,reserved, fac_commoners,
[itm_sword_medieval_a, itm_fighting_axe, itm_leather_jerkin, itm_skullcap, itm_hide_boots],
str_9 | agi_9 | level(4), wp(80), knows_common, mercenary_face_1, mercenary_face_2],
```

作为一个试验，现在已经准备好把它放入游戏中了。先让我们编译一下来看看我们的工作有没有错误的地方。运行Run build_module.bat。

3.4 雇佣兵 这一部分在现在的版本里面已经没有用了。我们将试着增加new_troop 兵种到玩家的队伍队列里面。以后我也许会把他们增加到酒馆可招募的兵种里面。

保存你的进度，打开module_parties.py。让我们为你的冒险旅途增加一些新的伙伴。我们将给你5 个new_troop 兵种到你的队伍队列里。像这样编辑“main_party”：

```
("main_party", "Main Party", icon_player | pf_limit_members, no_menu, pt_none,
  fac_player_faction, 0, ai_bhvr_hold, 0, (17, 52.5), [(trp_player,1,0), (trp_new_troop,5,0)]),
```

我们增加了(trp_new_troop,5,0)到玩家的队伍队列（不要忘记加 ，号！）。保存你的操作，关闭module_parties，然后双击build_module.bat。如果编译结束没有出现任何错误，你会发现你的新兵种出现在你的队伍里。（你需要开新档来让新兵种出现，你同样需要开新档来让队伍以及其他对象的修改生效。）

出发，打上两仗，注意你现在能让你的新兵种升级到“regular fighters”，当他们攒到足够的经验值的时候。

恭喜！你现在知道怎么编写以及熟练操作普通兵种了。我们将在下面这一部分介绍英雄，商人以及其他NPC。

3.5 NPC们

打开module_troops.py，查看NPC。你在游戏中看到的多种商人和NPC 与常规兵种非常类似。把他们区分开的最重要的元素是tf_hero 标识；正是这个标识使Marnid (马尼德)and Borcha（么么茶）获得特别身份。

在游戏中遇到的任何一个NPC 都是英雄，即便商人也是如此。英雄和常规兵种的主要区别在于：

1）英雄是不可杀死的。他们的生命用百分比表示，你只能拥有每个英雄的一个个体，除非他们被错误或人为的克隆。即使是人为设计的，克隆英雄也不是一个好主意。

2）每个英雄占用一整个部队栏。
3）英雄在兵种元组中赋予出生地后，会正确地出现在场景中。
4）玩家被敌人打败后英雄仍跟着玩家——英雄不被敌人俘获——但是玩家可以像通常那样俘虏敌人的英雄。

正因为每个英雄只有一个样本，他们没有复数形式的兵种名。英雄元组的第三个字段因此和第二个完全一样。

英雄元组的一个例子：

```
["npc2","Marnid","Marnid", tf_hero|tf_unmoveable_in_party_window, 0,reserved,
fac_commoners,
[itm_linen_tunic,itm_hide_boots,itm_club],
str_7 | agi_7 | int_11 | cha_6 | level(1), wp(40), knows_merchant_npc | knows_trade_2 |
  knows_weapon_master_1 | knows_ironflesh_1 |knows_wound_treatment_1 |knows_athletics_2 |
  knows_first_aid_1 | knows_leadership_1,
  0x000000019d004001570b893712c8d28d00000000001dc8990000000000000000],
```

这里是我们的伙伴Marnid，游戏过程中的忠实伴侣。他在标识字段中用tf_hero 标记为英雄。他还有tf_unmoveable_in_party_window 标识。这意味着除非他成了敌人的俘虏，否则你不能把他扔进驻地或者交换给别的队伍。以前我们可以在快乐野猪酒馆的入口4 找到他，但是在新版的代码里面他会随机出现在各个酒馆（关于这个的代码会在后面解释：JIK）。他是战斗中的炮灰，但他的不错的交易技能是初期冒险的得力助手，并且他永远不会挂掉除非一些脚本事件把他移除。而且，你会发现Marnid 有着独一无二的容貌代码。用游戏内置的容貌编辑器设计容貌，然后为你的mod 抓取容貌代码是可以实现的；这将在教程的第七部分讨论到。

另一个值得一提要点是，在这个文件里面马尼德的兵种id 是——“npc2”——。我们必须用小写字母并且不能加空格来引用它。如果你尝试在引用一个id 时使用大写字母，Mod 系统将抛出错误。因此，我们必须使用id“trp_npc2”来引用“Marnid”。

我们现在可以创造自己的英雄了。复制Marnid 的元组，粘贴到“#Add Extra Quest NPCs Below this point”这行说明文字下面。你可以把它放在我们刚才创建的“new_troop”的上面，这样可以让我们修改过的地方都放在一起。就像早先的一些规定，如果可以的话把你编写的代码都放在一起是个不错的主意。而有些代码则必须放在MOD 文件的一些特定的地方。因为更新版本的M&B 的发行，马尼德的状态会有些不同，但是都和以下的差不多。注意我们改变了他的ID 和名字来让他变成我们打算增加的一个新NPC：

```
["npc17","Geoffrey","Geoffrey", tf_hero|tf_unmoveable_in_party_window, 0, reserved,
fac_commoners,
[itm_courtly_outfit,itm_hide_boots,itm_club],
def_attrib | level(6), wp(60), knows_trade_3 | knows_inventory_management_2 | knows_riding_2,
0x000000019d004001570b893712c8d28d00000000001dc8990000000000000000],
```

此例中，我们把新英雄的id 改成了npc17（跟随在原先native 剧本里面的最后一个npc——npc16 之后）名字改成了“Geoffrey”。保持给NPC 命名的习惯有助于帮助你辨认他们。如

果我们把他创建为一个像马尼德那样能加入你队伍的英雄，最好把他放在紧跟在16 个NPC 的地方。不过，我们打算把他作为一个即将创建的任务里面的“角色”，所以还是把他放在他现在的位置上吧。为了让Geoffrey 看上去醒目一点（已经有了马尼德，他们两个不应该看上去就像双胞胎）让我们把他的itm_linen_tunic 改成itm_courtly_outfit。

现在请自行修改之。

这时候，如果我们双击build_module.bat, mod 系统会正确的进行编译。我们的新元组和mod 系统看见的没有丝毫冲突。然后如果这样做，我们会碰到一个大问题——Geoffrey 会出现在什么地方变得完全随机，就像其它NPC 一样。所有的（招募）英雄都随机出现（在酒馆）——场景标识代码为0。（JIK 提示：需要进一步解释，也许你不想让你的NPC 随机出现在酒馆。你得知道这个机制是怎么运作的）为了解决这个问题，我们把Geoffrey 设为town_6 也就是Praven（帕拉汶）的入口1。Praven 靠近地图的中央，所以对于游戏开始时随机出现在某处，那里比较容易找到。

你的元组看起来应当像这样：

["npc17","Geoffrey","Geoffrey", tf_hero, scn_town_6_tavern|entry(1),reserved, fac_commoners,
[itm_courtly_outfit,itm_hide_boots,itm_club],
def_attrib|level(6),wp(60),knows_trade_3|knows_inventory_management_2|knows_riding_2,
0x000000019d004001570b893712c8d28d00000000001dc89900000000000000000],

任意给他一些新装备或属性技能，就像我们为“new_troop”所做的。然后双击build_module.bat, 启动游戏进去Praven 酒馆。如果一切顺利的话，你会找到站在酒馆的Geoffrey。过去找到他。

试图和Geoffrey 对话将得到一个备用回答（雇佣军要求加入你的队伍），因为他当前没有被指派任何对话。

这是文件相互关联的又一个例子。他使用该对白取决于他在module_troops.py 文件中的位置，被定义于module_constants.py 文件中。我们稍候将讨论这方面的内容。

我们现在先离开Geoffrey 一会儿，等到以后我们讨论制作scenes、module_quests 和 module_dialogs 的时候再回来。在结束这一章节之前我们另外增加一个将要创建的任务的中的“角色”。搜索constable_hareck（治安官哈瑞克）。他的元组看起来像这样：

["constable_hareck","Constable Hareck","Constable Hareck", tf_hero, cn_zendar_center | entry(5),
reserved, fac_commoners,
[itm_leather_jacket,itm_hide_boots],
def_attrib|level(5), wp(20), knows_common, 0x00000000000c41c001fb15234eb6dd3f],

首先我们把他拷贝到Geoffrey 的下面。这让我们需要修改他的时候比较容易找到。因为他现在已经不会在游戏中出现，我们可以任意修改他。随后我们将给他一张新的面容（他的面容代码是旧版的）并把他放置在一个场景中。而现在我们只要记得他将会是我们即将建立的新任务的一部分。我们将把他们两个都放置到我们的MOD 里面的城镇酒馆种，但是在我们能那么做之前我们需要先创建场景和增加入口点。让们先让他拥有一个独一无二的ID：“hareck”，然后把他的场景代码改成0。

["hareck","Constable Hareck","Constable Hareck",tf_hero, 0,reserved, fac_commoners,

[itm_leather_jacket,itm_hide_boots], def_attrib|level(5), wp(20), knows_common,
0x00000000000c41c001fb15234eb6dd3f],

3.6 商人——这个在Winter 的指南中同样暂停了，仍然可以和商人交互

商人是英雄的特殊类型。除tf_hero 外，他们还有tf_is_merchant 标识。这个标识让他们除了原先在兵种元组中定义好的装备外，不装备物品库中的任何物品。换句话说，这些商人可以获得游戏过程中的所有物品，但不能够装备或使用之，这些物品很大可能是用来出售的。

商人的例子：

["zendar_weaponsmith","Dunga","Dunga",tf_hero|tf_is_merchant, scn_zendar_center|entry(3),0,
fac_commoners,
[itm_linen_tunic,itm_nomad_boots],
def_attrib|level(2), wp(20), knows_inventory_management_10,
0x00000000000021c401f545a49b6eb2bc],

这是Zendar 的武器商，名叫 Dunga。在原版M&B 中他非常貌似其他商人。如果你凑近看，唯一的区别是他们的id，名字，场景放置点和脸。

再次注意：这里没有测试过。建议将新的商人添加在“merchants_end”元组之前。

添加一个商人稍有点复杂。他们由于这个原因聚在一个组：对于每一类商人，M&B 中有脚本来每天更新他们的物品库——为此，这些脚本使用了“范围”，即一些连续的元组，从选择的开始点（下限）到结束点（上限）。例如，防具商的范围包括"zendar_armorer" (下限)到（不包括）"zendar_weaponsmith" (上限) 的所有东东。范围的上限并不包括在范围内，所以上限需要设置到条目的下一个(到 "zendar_weaponsmith")，如果我们想要防具商的范围包括"town_14_armorer"的话。

由于这个原因，新的防具商必须加在"zendar_weaponsmith"之前，新武器商必须加在“zendar_tavernkeeper"之前，新杂货商必须加在 "merchants_end"之前。

3.7 箱子(chest)

箱子兵种是特殊兵种，作为物品栏和玩家在游戏中交互。这些箱子兵种的物品栏是箱子，其本身并非箱子。

箱子，正如你游戏中见到的，一部分是场景道具，一部分是信息，一部分是兵种，一部分是硬编码的。新箱子创建起来有些复杂，遍布在不同的mod 文件中；这里我们只讨论 module_troops 中的相关信息。

箱子的例子：

["zendar_chest","zendar_chest","zendar_chest",tf_hero|tf_inactive, 0,reserved, fac_vaegirs, [],
def_attrib|level(18), wp(60), knows_common, 0],

所有箱子必须遵循这个例子。关于新箱子兵种，你唯一可以考虑改变的是兵种名字和id，（有可能）兵种级别和技能，以及物品库。上面提到，箱子兵种作为箱子的物品库存在于游戏内；因此，游戏开始的时候，你赋予箱子兵种的任何物品将出现在箱子中。

箱子都需要一个箱子兵种来发挥作用，同时它们也需要对不同的mod 文件做一些其他修改，关于这些我们在这些文件各自的文档中会涉及到。

学完这些之后，你已经掌握了所有了解module_troops 所必须了解的内容。至于其他你能够创造的兵种，可以在header_troops.py 中找到其标识列表。自由实验吧，当你准备好后，请继续教程的下一部分。

（Hetairoi 注：由于“箱子兵种”是一种只要一“对话”就打开对方物品栏的特殊兵种，对于其属性，实在是没有什么可以修改的地方，不过你可以给“箱子兵种”的技能里面添加knows_inventory_management_n，效果是增加箱子的容量，如果加到knows_inventory_management_10，就是把箱子的容量加到最大。）

第四部分

教程的第二部分中，我们学习了如何创造新队伍，即地图上唯一的位置。我们这里要讨论的是队伍模板(party template)，不要把它们和队伍相混淆。

简单而言，队伍模板是队伍出生在地图上时遵守的一组策略。下面这句话说明了队伍和队伍模板的最大区别——队伍是地图上的唯一实体，而模板并不实际存在于游戏世界。它们只是作为队伍出生时的一组策略。

因此，应该输入队伍id 的地方如果输入了队伍模板id，这些操作将失效。

从模板中生成的队伍不必是唯一的。同一个模板中可以有许多队伍；每个队伍会有一个随机数量的部队，这个数字取决于玩家的等级和模板中定义的最小/最大部队数量。

4.1 Module_Party_Templates 解析

该文件的开头是常见的Python 列表: party_templates = [, 紧接着是是几个跟游戏联系紧密的模板，这些模板不可编辑。

你会发现module_party_templates 中的元组和module_parties 中的相当类似，但这两者不可互换。

队伍模板的例子：

("village_farmers","Village Farmers", icon_peasant, 0, fac_innocents, merchant_personality, [(trp_farmer,5,10), (trp_peasant_woman,3,8)]),

这是我们都能在游戏中遇到的模板。此模板的队伍叫做“Village Farmers”（村民），你碰到他们时会自动开始对话，在游戏中他们表现得很软弱，每个队伍由农民（数量5 到10 个）和农妇（数量3 到8 个）两个部队栏组成。

（他们貌似已经废除了*pf_auto_start_dialog* 标识，也许事实上这个标识对所有部队都不能用了？）

元组字段解析：

1）元组模板id。用于在其他文件中引用这个元组模板。
2）元组模板名。该模板的队伍将使用的名字。
3）队伍标识。查看header_parties.py 列出的可用标识。
4）菜单。module_parties 文件中已摈弃它。在这里用0 值。
5）阵营。
6）个性(personality)。这个字段包括了队伍在地图上行为的标识。
7）栏列表。每个栏是一个包含以下字段的元组：
   7.1) 部队id。
   7.2) 栏内的最小部队数量。

7.3) 栏内的最大部队数量。
7.4) 成员标识（ 可选）。你需要添加一个额外字段来设置成员标识。例如(trp_swadian_crossbowman,5,14,pmf_is_prisoner)

队伍模板中最多可以有6 个栏。观察村民元组：

1）元组模板id = "village_farmers"
2）元组模板名= "Village Farmers"
3）队伍标识= icon_peasant
4）菜单= 0
5）阵营= fac_innocents
6）个性= merchant_personality
7）栏列表：
7.1) 部队id = trp_farmer，trp_peasant_woman
7.2) 栏内的最小部队数量= 5, 10
7.3) 栏内的最大部队数量= 3, 8
7.4) 成员标识（可选）=未设置。

如果你从第一部分开始学习的本教程，到这里你应当能相当熟练地阅读元组了，你会发现此元组中有一个字段与我们之前遇见的其他字段不太相同：字段6，个性标识字段。下一节我们会讨论这个字段及其功能。

4.2 个性（Personality）

正如在元组解析中提到的，个性字段决定了队伍在地图上的行为。这里你可以把自定义的分值赋给勇气值和好斗值，或者套用预定义的个性值，如merchant_personality（商人个性）。这些预设值是常量，包括勇气和好斗的分值。header_parties.py 中定义了所有预设值，所以现在打开这个文件并滚动到底部来看这些定义，你也会看到一列勇气和好斗的可能设置。

merchant_personality 常量在文件中的很多模板中都被用到。具有这个个性的队伍是友好的，不会出去攻击敌人或袭击弱旅。这是由于merchant_personality 把队伍的好斗设成了aggresiveness_0 。好斗为aggresiveness_0 的队伍永远不会攻击其他队伍。普通战斗队伍的预设是soldier_personality（士兵个性），具有好斗值aggresiveness_8，这使他们攻击敌对阵营的队伍，除非他们的兵力过少。

勇气是决定队伍何时逃离另一个更大队伍的数值。高勇气值意味着他们兵力劣势时不会很快逃离。

merchant_personality 的队伍勇气值为8，soldier_personality 为11。

这些设置值从0 到15，允许你精确地为队伍模板设置期望的行为。mod 制作新手最好坚持使用预设值。你的第一个mod 中需要用到的个性数值它们都包括了。

最后，对于强盗模板，有一个banditness(强盗）标识。这促使强盗队伍始终视附近的队伍为鱼腩，如果这个鱼腩带着数量可观的钱和/或商品，则会被强盗队伍攻击。理想的情况下，强盗队伍好斗指数低或兵力不足，以至于他不会攻击士兵队伍。注意：在这里你看不到强盗标识，而是在headers_parties.py 文件里列为强盗个性（bandit_personality）的一部分。

4.3 创建新模板

复制村民元组并粘贴到文件的最后，后括号之前。

("new_template","new_template",icon_peasant,0, fac_innocents,merchant_personality,
[(trp_farmer,5,10), (trp_peasant_woman,3,8)]),

这个例子中我们将id 从“village_farmers”改为“new_template”，名字也是同样。一旦做完这件事，就可以来编辑这个模板的细节了。

首先调整模板的阵营为fac_neutral（中立），把merchant_personality 改为soldier_personality。

("new_template","new_template",icon_peasant,0,fac_neutral, soldier_personality,
[(trp_farmer, 5, 10), (trp_peasant_woman,3,8)]),

现在开始，这个模板的队伍将是中立派，当看到敌人时会出击。下面我们来玩玩兵种组合。

("new_template","new_template",icon_peasant,0,fac_neutral,soldier_personality,
[(trp_npc17,1,1), (trp_new_troop,2,9)]),

这个例子有一个显著变化——最重要的一点是，我们在教程第三部分创建的英雄兵种trp_npc17（Geoffrey）现在领导这个队伍。由于不可能有一个以上的Geoffrey，我们将部队数量的最小最大值都设为1。

我们把第三部分中创建的常规兵trp_new_troop 分配给他作为随军。这个队伍中的"new troops"数量永远不会低于2，但随着玩家的等级提升，这个数目将逐渐调整。最终这个"new_template"模板可以生成9 数量的"new troops"——但是不会超过9。

保存文件并执行build_module.bat。如果一切顺利，就可以在mod 代码中使用这个新模板了。但是依赖队伍模板的队伍必须被生成出来，因为它们不会自说自话地出现在游戏中。如果我们现在启动游戏，将不会看到我们的"new template"生成的队伍在游戏中到处跑。

在以后的教程中我们将学习如何生成模板的队伍。此刻让我们离开Geoffrey 和他的小部队，在教程下一部分学习如何创建新物品吧。请继续看第五部分。

第五部分

第三、四部分中，我们学习了如何创建和装备新兵种和队伍模板。这个前提下，我们现在可以学怎样为部队创建新物品(item)。

5.1 Module_Items解析

module_items.py 的开头部分是一些常量，用来管理物品的修饰语(modifier)，修饰语是在游戏中调节物品用的。弯曲的长杆，有缺口的剑，重斧，这些都是由module_items 的元组创建出来的，然后给它们一些适当的物品修饰语以调高或调低物品属性。

这里定义的常量由标准物品修饰语构成。你在商人那儿和战利品中找到的物品会从这些常量中随机添加修饰符。商人物品栏和战利品中的物品不会使用物品修饰符常量中没有列出的修饰符。

对于经验老道的mod 制作者(modder)来说，下面这个事实非常有趣，即没有在相应常量中列出的修饰符仍然可以通过 "troop_add_item" 操作来使用。举个例子，长弓通常只有"plain"（普通的），"bent"（弯曲的）和"cracked"（破裂的）三种形式，但是如果我们给这个长弓加上修饰符 "balanced" （平衡的），并将其加入到玩家的物品栏，玩家就可以拿到一个平衡的长弓。

常量之后是元组的列表。我们感兴趣的第一个武器是"practice_sword"（练习剑），这是一个很好的范例。

我们来看一下这个元组：

["practice_sword","Practice Sword", [("practice_sword",0)],
itp_type_one_handed_wpn | itp_primary | itp_secondary | itp_wooden_parry | itp_wooden_attack,
itc_longsword, 3, weight(1.5) | spd_rtng(103) | weapon_length(90) | swing_damage(16,blunt) |
thrust_damage(10,blunt), imodbits_none],

这是一个在竞技场格斗和训练场中使用广泛的基本练习武器。元组字段解析：

1）物品id。用来在其他文件中引用物品。
2）物品名。出现在物品栏中的物品名字。
3）网格(mesh)列表。网格是包含以下字段的元组：
3.1) 网格名。游戏或mod 资源文件中的3d 模型名。
3.2) 此网格对应的修饰语位(bit)。用此网格来代替默认网格的物品修饰语列表。列出的第一个网格名为默认的（即普通属性物品的3d 模型名）。
4）物品标识。
5）物品性能(capability)。这个字段包括了物品使用的一系列动画(animation)。
6）物品价值。以第纳尔(denar)表示的基准值。注意：除非玩家的交易技能为10，物品在游戏中的实际价格将比这个基准值高许多。
7）物品属性。这里定义了物品的参数值：重量、充裕度、难度、护甲等级等等。
8）前缀位(bit)。物品可供使用的前缀。在module_items.py 文件的开头部分列出。

9) [可选]触发器。与该物品关联的一组简单触发器。

观察Practice_sword(练习剑)元组：

1) 物品id = "practice_sword"
2) 物品名 = "practice_sword"
3) 网格(mesh)列表：
3.1) 网格名 = "practice_sword"
3.2) 修饰语位 = 0
4) 物品标识 = itp_type_one_handed_wpn | itp_primary | itp_secondary | itp_wooden_parry | itp_wooden_attack
5) 物品性能 = itc_longsword
6) 物品价值 = 3
7) 物品属性 = weight(1.5) | spd_rtng(103) | weapon_length(90) | swing_damage(16,blunt) | thrust_damage(10,blunt)
8) 修饰语位 = imodbits_none
9) 触发器 =未设置。

从这个元组中我们可以知道“practice_sword” 的所有细节：

- 它使用“practice_sword” 网格作为默认网格。
- 它使用的物品标识使之成为单手近战武器。装备"practice_sword"的部队从装备库中选择武器时将视"practice_sword"为主要近战武器，同时他们选择后备（次要）武器时也会考虑"practice_sword"。（注意：目前次要武器的功能是含糊的，还不清楚它何时使用到甚至是否使用到。近战部队在战斗时肯定不会切换不同的近战武器）。我还没有发现*wooded_parry* 和*wooden_attack* 的意义。（Hetairoi 注，因为代表了被击打和击打时所引用的声效）
- 它具有itc_longsword 常量中定义的所有动画，所以使用"practice_sword"和使用长剑一样。
- 它的重量为1.5 千克，速度是103，武器长度是90。挥舞时的基准钝(blunt)器伤害为16，直刺时的基准钝器伤害为10。
- 它没有修饰语。

5.2 伤害类型

正如在元组中看到的，"practice_sword"发挥钝器伤害。显然这是因为它是练习武器，不过这却促使我们进一步看看M&B 中的伤害类型。

首先是砍**(cut)**伤。砍伤代表着利刃的切割动作，如剑或斧。砍伤对没有护甲或轻甲的敌人有伤害奖励，但对重甲却有很大的伤害惩罚。砍伤若使敌人生命值减到0，则会杀死他。

下面是钝伤。钝伤代表着无刃武器的钝击效果，如棒类和锤类。钝伤对重甲有50%的加成，但钝器通常比砍伤类武器短，总体伤害较小。钝伤的最大好处是在敌人生命值到0 时能够被击昏，而是被杀死。击昏的敌人可被俘虏和作为奴隶出卖。冲刺的马也可以发挥钝伤。

最后，刺伤代表着箭、弩箭和类似武器的穿刺。刺伤对重甲有50%加成，但出于平衡其他武器类型的目的，刺伤通常总体伤害较小。刺伤若使敌人生命值减到0，则会杀死他。

5.3 创建物品

把“practice_sword” 元组复制粘贴到文件底部，后括号之前。完成后把这个新元组的名字和id 改为“new_mace”。

["new_mace","New Mace", [("practice_sword",0)],
itp_type_one_handed_wpn|itp_primary|itp_secondary|itp_wooden_parry|itp_wooden_attack,
itc_longsword, 3, weight(1.5) | spd_rtng(103) | weapon_length(90) | swing_damage(16,blunt) |
thrust_damage(10,blunt), imodbits_none],

M&B 物品系统的灵活性相当好，只要几个微小的调节就能使剑变为钉头棒。就这个"practice_sword"来说，它已经被设为钝伤，这使得我们的工作更简单了。

首先，我们把新钉头棒的物品性能从itc_longsword 改成itc_scimitar（弯刀）。这使得我们的钉头棒失去了刺的能力，因为M&B 的itc_scimitar 常量并没有包含直刺的动画。接下来把物品的网格由"practice_sword"改成"mace_pear"。这是原版中没有用到的网格，所以这一步将使我们的新钉头棒具有崭新的模样。我们同样也去掉“itp_wooden_attack”，因为其它锤类都没有这个标识。

["new_mace","New Mace", [("mace_pear",0)],
itp_type_one_handed_wpn|itp_primary|itp_secondary|itp_wooden_parry,
itc_scimitar, 3, weight(1.5) | spd_rtng(103) | weapon_length(90) | swing_damage(16,blunt) |
thrust_damage(10,blunt), imodbits_mace],

这个例子中，我们按计划改变了网格，并把修饰语位从imodbits_none 改成了imodbits_mace。这将让我们的新钉头棒具有文件头部imodbits_mace 常量中定义的所有修饰语。

只要另外的两处修改我们就可以完成这个物品了。特别地，我们将要改变其砍伤，以及给它一个额外的物品标识。

观察：

["new_mace","Geoffreys mace", [("mace_pear",0)] ,
itp_type_one_handed_wpn|itp_primary|itp_secondary|itp_wooden_parry|itp_unique, itc_scimitar,
3,weight(1.5)|spd_rtng(103)|weapon_length(90)|swing_damage(26,blunt)|thrust_damage(10,blunt)
,imodbits_mace],

正如你所见，我们把砍伤由16 调到26，这使得我们的新钉头棒在战斗中更具危险性。特别地，我们给标识字段加入了itp_unique 标识。具有itp_unique 标识的物品不会在普通的战后战利品栏中出现。由于我们对这个钉头棒有其他打算，这个标识可以让玩家不会过早的得到它。

最后一步调节是把物品名从“New Mace”改为“Geoffreys_mace”。然后，打开module_trooops.py 并把Geoffrey的物品库里的itm_club 改成itm_new_mace。

保存所有的文件，执行build_module.bat。你将会注意到有个提示”itm_new_mace”没有定义：

这个说明了执行编译的次序可能和我们编写时的次序不一样。说明在编译module_items.py之前先编译了module_troops.py。因为如此，我们的new mace 代码此时还没有被看到。如果我们再次运行build_module.bat，这个错误就不会再出现，因为module_items.py 在前面的编译过程中出现错误了之后仍然会被编译。这再次提醒我们你应该在对MOD 文件有了任何修改之后经常进行编译，特别是你增加了某些别的模块即将需要用到的东东。

恭喜！你已经制造了一个崭新的物品，而且将其加入了兵种的装备库中。这个兵种是一个英雄，所以他的装备库中总是会有这个新钉头棒，而且总是会使用装备库中能够使用的最佳武器。

另一方面，常规兵种是从装备库中随机选择装备。正因为如此，大多数常规兵穿着各式各样的装备——否则他们会看起来一个样。

现在我们知道了如何创建新物品，可以看一下变化多端的属性值和研究它们的意思了。

5.4 物品属性(Stat)

这一节中你会看到全面的属性列表和其功能的明细。由于一些属性对不同物品类型有着不同意义，我们按照物品类型把列表归了类。

General（综合类）

abundance（充裕度）——百分比。

这个属性决定了物品出现在商人物品栏和战利品栏中的频率。100 是基准，可以大于或小于100（最小是0），而100 的充裕度并不意味着这个物品铁定会出现。现在我还不知道让物品铁定出现的最高比例是多少。

weight（重量）——千克。

用千克数定义了物品的重量。

itp_type_horse（马匹类）

body_armor（护甲）——值。

决定了马匹的护甲值和生命值。值越大意味着护甲和生命值越多。

difficulty（难度）——值。

决定了玩家骑这匹马所需要的骑术。

horse_speed（马匹速度）——值。

战场上的马匹速度。值越大马越快。

horse_maneuver（马匹操控）——值。

战场上的马匹灵活性。

horse_charge（马匹冲刺）——值。

决定了马匹撞上步兵时的输出伤害，以及冲撞后损失的速度。值越大，马给出的伤害越高，能撞开越多人数的步兵堆。

itp_type_one_handed_wpn（单手武器类）

difficulty（难度）——值。

使用该武器所需的最小力量(STR)值。如果力量达不到该值，就不能够使用它。

spd_rtng（速度）——值。

武器的挥砍和直刺攻击速度。

weapon_length（武器长度）——厘米。

厘米表示的武器长度。这个属性决定了游戏中武器可以够到的范围，与网格的大小无关。

swing_damage（砍伤）——值，伤害类型。
武器挥砍攻击时的基准伤害和伤害类型。
thrust_damage（刺伤）——值，伤害类型。
武器直刺攻击时的基准伤害和伤害类型。

itp_type_two_handed_wpn（双手武器类）
同itp_type_one_handed_wpn。

itp_type_polearm（长杆武器类）
同itp_type_one_handed_wpn。

itp_type_arrows（弓箭类）
weapon_length（武器长度）——厘米。
厘米表示的弓箭长度。
thrust_damage（刺伤）——值，伤害类型。
此类箭在弓的基准伤害之外的附加伤害，以及伤害类型。
max_ammo（最大箭数）——值。
每袋箭的数量。

itp_type_bolts（弩箭类）
同itp_type_arrows。

itp_type_shield（盾类）
hit_points（生命值）——值。
盾的基准生命值。
body_armor（护甲）——值。
盾每次被击到的伤害减免。
spd_rtng（速度）——值。
盾切换到防御模式的速度。
weapon_length（武器长度）——值。
盾的覆盖度。值越大，身体被盾覆盖的范围越大，保护身体免受弓箭伤害的作用越大。

itp_type_bow（弓类）
difficulty（难度）——值。
使用该弓所需的最小强弓技能点数。如果兵种没有达到该强弓技能点数，将不能装备它。
spd_rtng（速度）——值。
弓的装填速度。也就是说，弓弦颤动、装箭、再次拉弓的迅速程度。值越大装填的速度越快。
shoot_speed（射速）——值。
从该弓射出的箭的速度。值越大箭越快；注意，太快的箭可能会毫发无伤地穿过近距离的敌人。
thrust_damage（刺伤）——值，伤害类型。
弓输出的基准伤害和伤害类型。
accuracy（准确度）——百分比。
射出点刚好和瞄准点重合的几率。100 意味着可能性为100%，较低的值会严重降低射中的几率。原版中的
弓和弩未使用这个属性，但是可以添加之。

itp_type_crossbow（弩类）
difficulty（难度）——值。
使用该弩所需的最小力量(STR)值。如果力量达不到该值，就不能够使用它。

spd_rtng（速度）——值。
弩的装填速度。也就是说，弩弦颤动、装箭、再次拉弦的迅速程度。值越大装填的速度越快。
shoot_speed（射速）——值。
从该弩射出的箭的速度。值越大箭越快；注意，太快的箭可能会毫发无伤地穿过近距离的敌人。
thrust_damage（刺伤）——值，伤害类型。
弩输出的基准伤害和伤害类型。
max_ammo（最大箭数）——值。
再次装填之前可以射出的箭数。
accuracy（准确度）——百分比。
射出点刚好和瞄准点重合的几率。100 意味着可能性为100%，较低的值会严重降低射中的几率。原版中的弓和弩未使用这个属性，但是可以添加之。

itp_type_thrown（投掷类）
difficulty（难度）——值。
使用该武器所需的最小强掷技能点数。如果没有达到该强掷技能点数，将不能装备它。
spd_rtng（速度）——值。
武器的装填速度。即准备下一个投掷物的迅速程度。
shoot_speed（射速）——值。
抛掷物在空气中运动的速度。
thrust_damage（刺伤）——值，伤害类型。
武器输出的基准伤害和伤害类型。
max_ammo（最大弹药数）——值。
每个格子的武器（即投掷物）数量。
weapon_length（武器长度）——值。
厘米表示的武器长度。

itp_type_goods（杂货类）
food_quality（食物品质）
食物对队伍士气的影响。高于50 的食物被消费时会提高士气，低于50 的会降低士气。
max_ammo（最大数量）
物品的可消费数量。

itp_type_head_armor（头盔类）
head_armor（护头）——值。
头盔保护头部免受伤害的数值。
body_armor（护身）——值。
头盔保护身体免受伤害的数值。
leg_armor（护腿）——值。
头盔保护腿部免受伤害的数值。
difficulty（难度）——值。
装备头盔所需的最小力量值。

itp_type_body_armor（护甲类）
同itp_type_head_armor。

itp_type_foot_armor（靴子类）
同itp_type_head_armor。

itp_type_hand_armor（臂铠类）

同itp_type_head_armor。

itp_type_pistol（手枪类）

difficulty（难度）——值。

装备枪械所需的最小力量值。

spd_rtng（速度）——值。

枪械的装填速度。即重新装填弹药并瞄准的速度。

shoot_speed（射速）——值。

从该枪械射出的子弹的速度。

thrust_damage（刺伤）——值，伤害类型。

枪械输出的基准伤害和伤害类型。

max_ammo（最大箭数）——值。

再次装填之前可以射出的子弹数量。

accuracy（准确度）——百分比。

射出点刚好和瞄准点重合的几率。100 意味着可能性为100%，较低的值会严重降低射中的几率。

itp_type_musket（步枪类）

同itp_type_pistol。

itp_type_bullets（子弹类）

同itp_type_arrows。

第六部分

在文档的这个部分，我们考察MOD 系统里面最小的那些文件，也是对你的MOD 里其它文件影响最小的文件。不过，这些文件仍然可以通过多种方法提供用处。通过这一章节，你将学会它们的所有组成。

6.1 Module_Strings解析

module_strings.py 是系统中最简单的文件。它包含了字符串——用于在各处显示的文本段。在MOD 系统中各处都使用字符串，从module_dialogs 到module_quests；只要是想在屏幕上显示一段文字的地方。字符串模块，存储了独立的字符串，不受单独文件的（使用）约束。因此它们可以被整个MOD 系统的任何操作所调用。

在游戏中，你可以看到这些字符串被用在了种种地方，如屏幕左下白色的滚动文字，或者在指南对话框中，甚者插入游戏菜单或者人物对话中。

类似于module_factions，这个文件直接开始于一个Python 列：strings = [。同样的这个文件的最初几个元组是游戏内置的不能被修改。

字符串的例子：

("bandits_eliminated_by_another", "The troublesome bandits have been eliminated by another party."),

或者

("s5_s_party", "{s5}'s Party"),

元组解析：

1）字符串ID。用于在其它文件中引用字符串。
2）字符串文本。

字符串的一个值得注意的功能就是可以在其内部放置寄存器值或者干脆就是另一个字符串。你可以通过给字符串加上诸如{reg0}一类的来实现。那样将会显示reg(0)的当前值。而{reg10}将显示reg(10)的当前值，诸如此类。

作为额外的字符串，你必须使用字符串寄存器而不是普通寄存器，这是因为字符串是由寄存器单独存储的。

例如，{s2}将显示字符串寄存器的内容——而不是reg(2)的内容。你可以自由地使用字符串寄存器2 和reg(2) 同时存放不同的内容，它们并不会重叠或者相互影响。

在对话中，也可以显示由说话者的性别来区别的某个字符串的部分（或全部）内容。例如，插入{sir/madam}到一个字符串中将使得当说话者为男性时显示“sir”，而如果是女性则显示“madam”。

所有的选项（除了按性别替换的）将会在任何字符串区域自如地发挥作用。按性别替换的只

在对话中起作用。

作为实现“显示信息”这个功能的目的，通过在选项中附加十六进制代码，可以以多种颜色显示一条字符串。例如，

(display_message,<string_id>,[hex_colour_code]),

十六进制颜色代码按照RGB 格式制定，例如（示例为白色= 0xFFFFFFFF）：

0xFF	FF	FF	FF
不能确认这是什么，但是所有颜色从这个开始（foxyman注：我猜测这个是alpha，如果你把它改小，应该会变半透明效果）	红色部分，从00（0）到FF（256）	蓝色部分，从00（0）到FF（256）	绿色部分，从00（0）到FF（256）

所以要显示红色你需要使用代码0xFFFF0000，这个代码里面红色值为256，蓝色为0，绿色也是0。知道如何调色将在你需要标准色彩以外的颜色时有所帮助。

6.2 Module_Factions解析

module_factions.py 包含了所有MOD 系统和M&B 人工智能使用的势力。我们注意这个小文件负责的如下几点：

文件由“factions = [”开始，和多数MOD 文件一样，前面一小段属于系统指令，不要修改。如果你向下拉一点，你会找到“deserters”元组。作为一个势力的例子：

("deserters","Deserters", 0, 0.5,[("manhunters",-0.6),("merchants",-0.5),("player_faction",-0.1)], [], 0x888888),

这是一个短小、简单的元组。它管理着“deserters（乱军）”势力，它们的主要敌人是“manhunters（赏金猎人）”，“merchants（商队）”，和“player_faction（玩家）”势力。

如果一个势力在MOD 常量的任何地方都没有定义与“deserters”的关系值，就会自动置为0，因此这两个势力将彼此相安无事。

元组各部分解析：

1）势力ID：用于在其他文件中对势力进行引用。
2）势力名称
3）势力标记
4）势力一致性：一个势力中各个成员的关系。
5）势力间关系：每个关系是一个段，包含如下内容：
   5.1）势力：相关势力名。
   5.2）关系：两势力间关系，取值范围-1（坏）~1（好）。
6) 等级
7) 势力颜色（默认为灰白色）

查看“deserters”元组：

1）势力ID = " deserters "
2）势力名称= "deserters"
3）势力标记= 0
4）势力一致性= 0.5
5）势力间关系：
5.1）势力= "manhunters", “merchants”, "player_faction"
5.2）关系= -0.6, -0.5, -0.1
6) 等级未列出（JIK 提示：尚不知道此处含义）
7) 势力颜色为16 进制值表示的0x888888 （类似于字符串，只有最后6 位为RGB 颜色代码）

这个势力没有标记，势力凝聚力为中，定义了3 个敌对势力。可是，如果我们看看其他地方，我们可以发现（例如）所有“kingdom_#”势力和“deserters”势力的关系为-0.02，这个定义在所有的王国势力元组里面（例如“kingdom_1”，“kingdom of Swadia（芮尔典王国）”）。

这是由于关系对双方都起作用——它只需要设立一次，对双方势力都起作用。

现在，拷贝“deserters”创建一个新势力，粘贴在列表最后。将势力ID 和名称改为“geoffrey”，删除关系列表中的所有势力对象除了“player_faction”，将其值改为-1。保存文件，点击build_module.bat，只是这次在我们修改了其它地方之前不会编译。我们将使用这个势力在别的mod 文件里面做定义。保存。完成上面的工作之后，打开 module_party_templates.py 和module_troops.py，改变“new_template”和部队“npc17”的势力为“fac_geoffrey”。保存，关闭文件。点击build_module.bat。如果一切正常，可以看出这两个部队的势力变为了"Geoffrey"，且对玩家怀有敌意。

6.3 Module_Constants解析

module_constants.py 是一个非常简单的文件。它由常量组成。经由阅读前三章的内容，你应该知道它们是如何工作的。在常量模块中的常量和定义在其他地方的常量完全相同，但是用在多个模块文件中的常量被定义在常量模块中以使MOD 系统能够找到它们。

常量模块同时作为一个有组织的，易理解的列表，在其中你可以在你需要时改变常量的值。任何改动将会立刻影响所有使用此常量的操作，你可以不用手工寻找/改动。

例如，修改hero_escape_after_defeat_chance = 80 为hero_escape_after_defeat_chance = 50 将立即改变人和使用到hero_escape_after_defeat_chance,这个常量的任何地方，将这个常量视同数值50 而不再是80。

这个文件同样还记载了范围标记例如kings_begin = "trp_kingdom_1_lord" and kings_end = "trp_knight_1_1"。

6.3a 对象槽（Slot）（JIK 提示：需要深入了解槽的概念）

另一个定义在常量部分的强大工具是槽（slot）的运用。槽可以分配给任何元组对象（兵种（troops），势力（factions），部队（parties），人物（quest），等等）并且对于所有的特定对象类型可以赋予一个特殊的值。让我看看任务的槽是在哪里列出来的：

```
############################################################
## QUEST SLOTS ##############################
############################################################
slot_quest_target_center = 1
slot_quest_target_troop = 2
slot_quest_target_faction = 3
slot_quest_object_troop = 4
##slot_quest_target_troop_is_prisoner = 5
slot_quest_giver_troop = 6
slot_quest_object_center = 7
slot_quest_target_party = 8
slot_quest_target_party_template = 9
slot_quest_target_amount = 10
slot_quest_current_state = 11
slot_quest_giver_center = 12
slot_quest_target_dna = 13
slot_quest_target_item = 14
slot_quest_object_faction = 15
slot_quest_convince_value = 19
slot_quest_importance = 20
slot_quest_xp_reward = 21
slot_quest_gold_reward = 22
slot_quest_expiration_days = 23
slot_quest_dont_give_again_period = 24
slot_quest_dont_give_again_remaining_days = 25
############################################################
```

在等号左边的是每个槽的名称。槽的号码在等号的右边。槽的名称可以易于理解该槽的用处。特别地，“slot_quest_xp_reward”是native 剧本中用来保存你完成任务后将得到多少经验值。当引用这个值时，你既可以通过slot_quest_xp_reward 也可以通过它的号码21 来引用。当查看代码的时候，使用槽的名称比号码更直观。

当我们创建任务时，我们需要使用任务的一些槽。通过这种方法，数值可以在一个地方进行设置，而当我们需要改变这个值的时候，我们只要在一个地方改变它。而另一个好处就是当玩家接手了多个任务的时候，我们不必担心也许会因此覆盖我们设置的变量。每个任务（就像所有的元组对象）都有自己的槽。

6.4 Module_Quests解析

module_quests.py 是最后一个小而简单的文件。它包含了任务，包括所有关于人物的文字。在这里添加新任务可以经由MOD 系统使之运作，因此进程能够读取任务现在的状态，并且将此状态作为一个条件操作。

这同样会定义一个任务对象，而其之前提及到的槽就能派上用场。一个任务的例子：

```
("deliver_message", "Deliver Message to {s13}", qf_random_quest,
"{s9} asked you to take a message to {s13}. {s13} was at {s4} when you were given this quest."),
```

这是一个普通的每个领主都会要你去做的跑腿任务。

段解析：

1）任务ID = "deliver_message"
2）任务名称= "Deliver Message"
3）任务标记= qf_random_quest
4）任务描述= "{s9} asked you to take a message to {s13}. {s13} was at {s4} when you were given this quest."

这个任务使用到字符串寄存器（{s9}，{s13}，和{s4}）来说明谁给你这个任务（{s9}），接收者（{s13}），以及接收者最后被看到出现在哪里（{s4}）。查看类似的任务，将会发现{s9}总是用作显示给你任务者的姓名，{s13}总是行动目标的个人或者部队，而{s4}是他们最后出现的位置。

如果你打算建立一个没有任何标识的任务，把标识区域置一个0 就可以了。

现在，我们已经初步建立了我们的小任务。拷贝粘贴下面一段在Python 列表底部，在最后一个条目("quests_end")的上面：

```
("mod_trouble", "Speak with the troublemakers in Mod Town", qf_random_quest,
"Constible Haleck has asked you to deal with a bunch of young nobles making trouble around Mod Town.\
How you want to tackle the problem is up to you; He has promised a small purse if you succeed."
),
```

注意这段的构造。你可以看到，只进行了一点修改就建立一个新任务，但是把它融入游戏中还有很多工作要做。注意：似乎所有这里列出的任务都有gf_random_quest flag 标示，这个也许是必须的。注意到我们使用了一个 \ 符号来分开长的字符串。这并不会使游戏中出现斜杠，只是让代码易读。

我们需要让你接到这个任务的地方，任务中的“演员”以及一些有趣的对话。先前我们创建了演员（Hareck 和Geoffrey）。下面我们将创建你接到这个任务以及碰到“行动目标”的“场景（scene）”。完了以后，我们下一个目标是module_dialogs.py——我们将在那里放置我们要使用的新任务，部队和队伍模版。

保存，关闭文件，点击build_module.bat。如果一切正常，我们现在可以进入第七部分。

第七部分

在我们制作自己的场景之前，让我们看看如何摆脱一些游戏测试当中次要的、麻烦的东西。这一部分会集中在整理菜单上，但首先这里有一个更简单的方法，让我们不需要漫无目地在卡拉迪亚游荡去我们要去的地方。游戏设定让你在五个训练场中的一个附近开始游戏。你也许知道怎么来使得经过游戏开始的选择后让固定地从哪个训练场开始游戏的方法，但是让我们修改一下，以便你可以从mod 里的城市开始游戏。如果你不想单独做这个东西，你可以把这段代码覆盖掉训练场的代码，但是由于新版本发布和代码的变化，最好是看懂我在这里做了些什么，然后自己来做：

```
("training_ground_1", "Training Field",
icon_training_ground|icon_training_ground|pf_hide_defenders|pf_is_static|pf_always_visible|pf_label_medium,
no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(-3,-3),[], 100),
# ("training_ground_1", "Training Field",
icon_training_ground|icon_training_ground|pf_hide_defenders|pf_is_static|pf_always_visible|pf_label_medium,
no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(44,61),[], 100),
# ("training_ground_2", "Training Field",
icon_training_ground|icon_training_ground|pf_hide_defenders|pf_is_static|pf_always_visible|pf_label_medium,
no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(-38, 81.7),[], 100),
# ("training_ground_3", "Training Field",
icon_training_ground|icon_training_ground|pf_hide_defenders|pf_is_static|pf_always_visible|pf_label_medium,
no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(89.2, 16),[], 100),
# ("training_ground_4", "Training Field",
icon_training_ground|icon_training_ground|pf_hide_defenders|pf_is_static|pf_always_visible|pf_label_medium,
no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(9.5 , -80.5),[], 100),
# ("training_ground_5", "Training Field",
icon_training_ground|icon_training_ground|pf_hide_defenders|pf_is_static|pf_always_visible|pf_label_medium,
no_menu, pt_none, fac_neutral,0,ai_bhvr_hold,0,(-34.75, -30.3),[], 100),
```

这是个快速的方法。我复制了第一个训练场条目，把它的位置改成(-3,-3)。这个位置靠近我的mod 中的位置是(-1, -1)的城市。然后我注解掉所有的元组。在做像这样的修改时，最好把mod 作出来并在游戏中测试。

这样的修改是次要的，但如果游戏中其他的语句出现问题，这可能是因为它1.没有构筑2.没有载入或3.在特殊情况下出了问题。如果你改变或删除了原版代码，最好测试一下。这些改动使人物只能从一个训练场附近出生，它距离我们的目标较近。如果你不想让训练场出现，你就可以把pf_disabled 这个标识加入到标识列表里。仍然会从训练场附近出生，但在游戏

里是不可见的。

7.1 快速创建角色(由MartinF 设计)

减缓测试进程之一的是角色的创建。这在游戏中很好，但会浪费不少测试mod 的时间。

我们可以预先设置好角色的数据，然后开始游戏。打开module_game_menus.py。让我们对第一个元组做一个快速的分析，“start_game_1”：

1) 游戏菜单ID （编译后自动在游戏菜单ID 前面加上前缀menu_） = “start_game_1”
2) 游戏菜单标识（整数）。
查看header_game_menus.py 列出的可用标识 =
menu_text_color(0xFF000000)|mnf_disable_all_keys
3) 游戏菜单文本（字符串） = "Welcome, adventurer, to Mount&Blade. Before…”
4) 网格名（字符串）。目前不使用，必须是字符串"none"
5) 执行块。当菜单被激活时的执行动作 = []
6) 菜单选项列表（列表）
   6.1) 菜单选项ID（字符串）用于在其它文件中关联游戏菜单。 = start_male and start_female
   6.2) 条件块（列表）。如果条件不符合，该菜单选项将不会被显示 = [] and []
   6.3) 菜单选项文本（字符串）。玩家选择选项时看到的可供点击的选项内容 = Male and Female
   6.4) 结果块（列表）。当选项被选后的执行动作。

我们要在选择男性或女性之间加一个新选项，你可以把这些复制到女性选项上面：

```
############## MF for testing start ####################
("start_mod",[(eq,1,1),],"Quick Character (for mod testing)",
[(troop_set_type,"trp_player",0),
(assign,"$character_gender",tf_male),
(troop_raise_attribute, "trp_player",ca_intelligence,-4), #so you dont need to wait time picking extra skills
#(troop_raise_skill, “trp_player”, skl_pathfinding, 5),
#(troop_raise_proficiency, "trp_player",wpt_crossbow,40),
#(party_add_members, "p_main_party", "trp_swadian_knight", 10),
#(troop_add_item, "trp_player","itm_sword_medieval_a", imod_rusty),
(change_screen_return, 0),
]
),
############## MF for testing end ####################
("start_female",[],"Female",
[
(troop_set_type,"trp_player",1),
```

你开始新游戏时，在男性和女性之间会有一个新选项：“Quick Character (for mod testing)”。这只有在情况合适的时候才是可见的。(eq,1,1)这个条件看起来奇怪，但它是一个开启或关

闭选项的简易方法。把它编辑到(eq,1,2)。这样一个小改变会使条件失效，菜单选项会变得不可视。你也许要在选项的文件头加上关于如何关闭选项的自己的注释。菜单选项非常直接明了，所以我们跳过它到操作这一块。

像普通的男性选项一样，我们模仿设定玩家为trp_player 的前两行。然后指定一个性别。要设定这两行。

然后我们根据需要设定。减少智力会使可分配的技能点减少，这让角色技能属性点数分配画面快速通过。如果你把4 个属性点（强制的）加到力量或魅力上，你在技能和武器熟练度上就不会有额外的点数。

下面几行我添加了注释，但是可以给你一些快捷的方法。如果你要测试技能和武器熟练度，可以使用这个。如果你要测试一个兵种或一件特殊武器，也可以使用这个。最后一行(change_screen_return,0)结束了开始部分，把你带到角色建立窗口。

你还可以更简单的测试游戏的功能。比如增加经验、金钱、部队和物品这样简单的东西，脚本或mission_templates 也可以在游戏菜单中简单的完成。普遍使用的是建立营地菜单。查找“You set up camp”。

然后可以在菜单中增加自己的选项：

```
("camp",mnf_scale_picture,
"You set up camp. What do you want to do?",
"none",
[(assign, "$g_player_icon_state", pis_normal),
(set_background_mesh, "mesh_pic_camp"),
],
[
("camp_modding",[],"Go to the modding menu.",
[(jump_to_menu, "mnu_camp_modding"),]
),
("camp_action_1",[(eq,"$cheat_mode",1)],"Cheat: Walk around.",
```

这个只把玩家带到一个新的菜单选项，我们现在建立那个菜单，我们把它建立在module_game_menus.py的最后：

```
################ MF for testing start ####################
("camp_modding",0,"Select an option:","none",[],
   [("camp_mod_1",[],"Increase player's renown.",
      [(str_store_string, s1, "@Player renown is increased by 100. "),
      (call_script, "script_change_troop_renown", "trp_player" ,100),
      (jump_to_menu, "mnu_camp_modding"),
   ]
),
### MF - change attributes and skills below, or add weapon proficiencies with
###(troop_raise_proficiency, "trp_player", wpt_). See header.troops.py for options
("camp_mod_2",[]"Raise player's attributes and skills.",
   [(troop_raise_attribute, "trp_player",ca_strength,20),
```

```
   (troop_raise_skill, "trp_player",skl_riding,10),
   (jump_to_menu, "mnu_camp_modding"),
### MF - Spawn any party you want near your party. Look in party_templates.py for pt_id
("camp_mod_3",[],"Spawn a party nearby",
   [(spawn_around_party, "p_main_party", "pt_looters"),
   (display_message, "@Party spawned nearby."),
   ]
),
("camp_mod_4",[],"Back to camp menu.",
   [(jump_to_menu, "mnu_camp"),
   ]
),
]
),
############### MF for testing start ####################
```

如你所见，这里能测试许多东西，我给了3 个例子。第一个增加声望。也可用于经验或金钱。下一个跟技能有关。这里增加了力量和骑术。如果你足够有创造力，你可以制作一个增加或减少各项属性的菜单。第三个是在玩家周围召唤部队。这对测试战斗或AI 响应很有用。

我们还能做出给你部队、伙伴、物品、开始对话和其他的东西。这对新手来说是一个很有用的功能。

你在这里所要掌握的关键就是你可以使用命令(jump_to_menu, “mnu_<你的菜单名>”)来跳到你想要的菜单。这个可以在任何运行模块中进行调用。（再次感谢MartinF 分享本章节）

第八部分

本部分的教程我们将为任务制作一个场景，这需要进入游戏并使用内置的编辑模式。我之前说过，我们也会用这个模式来给治安官哈瑞克一个新的脸型。不过在此之前，我们必须先做好Mod 城镇。到目前为止我们创建了它，并放置在了地图上，但还需要一些步骤来使其成为一个带场景的完整城镇。这部分我们可以在游戏中进行编辑。

8.1 增加一个新的场景项

让我们先从给我们的城镇定义一个酒馆开始。场景存储在module_scenes.py 文件中。在目前的状况下，一个随机的城镇场景会被用到这个城镇（如果你尝试进入Mod 城镇的话会注意到这一点）。我们来看看其中一个酒馆场景：

("town_7_tavern",sf_indoors, "interior_town_house_f", "bo_interior_town_house_f", (-100,-100), (100,100), -100, "0", ["exit"], []),

现在你应该已经对各项目间的分隔符很熟悉了（亲切的‘，’）。你也应该学会了在文件开头找到元组定义。

这里我们再次将module_scenes.py 中部分列出来：

1) 场景代号（Scene id）{字符串}：用于在其他文件中引用该场景。每个scene-id 会被自动加上前缀scn_。
2) 场景标识符（Scene flags）{整数}：可用的flag 参考header_scenes.py。
3) 网格模型名称（Mesh name）{字符串}：只用于室内场景。对于室外场景使用关键字"none"。
4) 碰撞模型名称（Body name）{字符串}：只用于室内场景。对于室外场景使用关键字"none"。
5) 最小坐标（Min-pos）{(浮点数（可以带小数的值），浮点数)}：(x,y)坐标的最小值。玩家不能移动到此
坐标以外。
6) 最大坐标（Max-pos）{(浮点数，浮点数)}：(x,y)坐标的最大值。玩家不能移动到此坐标以外。
7) 水面高度（Water-level）{浮点数}。
8) 地形代码（Terrain code）{字符串}：你可以从地形生成界面复制地形代码。
9) 从此场景可以到达场景的列表{字符串列表}。（已弃用，以后版本的Module System 可能会删除）（新的系统使用passages（通道）完成从一个场景转换到另一个场景；通过通道的variation-no（变量号）进入菜单中的相应项。）
10) 该场景使用的chest-troops（储物箱兵种）列表{字符串列表}。通过编辑模式放置这些储物箱使你可以在该场景访问储物箱。

储物箱将通过variation-no 在这个列表中选择具体访问哪个兵种的物品栏。

你也注意到了一些只在城镇中特有的定义。在修改一个模块之前，看看这些部分总是好的，

总能得到些有用的信息。这里我们注意到：“可用的flag 参考header_scenes.py”。

让我们复制一个"town_7_tavern"到所有酒馆的最后，紧接着"town_18_tavern"。然后作如下修改：

("town_mod_tavern",sf_indoors,"interior_town_house_f", "bo_interior_town_house_f", (-100,-100), (100,100), -100, "0", ["exit"], []),

没错，我们需要改的就只是一个代号而已。保持他跟城镇文件中的城镇顺序相同很重要，因为他们是按顺

序嵌入的。这样也就让它成为了Mod 城镇的一部分。我们用的是乌克斯豪尔的酒馆，因为是现成的，我们只需要加一些东西就可以了。运行build_module.bat。

如果你之前因为好奇访问过Mod 城镇的各个场景的话，你会发现你到达的场景和从菜单中选择的选项不一致。因为每个城镇（包括城堡和村庄）的场景是通过module_scripts.py 中的一个循环指定的。所有场景被按顺序分配给对应顺序的城镇。这就是为什么让所有内容保持一致的顺序很重要。如果你进入Mod 城镇的竞技场，你会看到town_1 的地牢，因为这是town_18_arena 的下一个内容。Mod 城镇是第19 个城镇，它会提取第18 个场景后面的那一个。

8.2 游戏中的场景编辑器

理想情况下编译会成功。在启动M&B 的时候，在选择剧本的界面点选设置，在高级选项卡下，勾选编辑模式。注意那个警告。我们会在做好场景之后关掉这个选项。不过只要不进入大规模的战斗，也不会有什么问题。在视频选项卡下，确保窗口模式被选定。最好将分辨率设为小于你的桌面分辨率。如果你搞不清楚，就设为800×600。点击确定并运行MOD。

既然到了这一步，让我们来给哈瑞克一个新的face code（脸型代码）。开始新游戏，在编辑脸型的界面，修改出你想给哈瑞克的脸型。做好之后，按ctrl-E。在脸部前面会出现一个窗口显示其face code。点选它复制到剪切板，并粘贴到别的地方留待后面再用。如果你搞丢了或者没兴趣花时间去编辑脸型，这里我做了一个：

0x0000000e8000558036db6db6dbefb6db00000000001db6db0000000000000000

进入游戏并进入Mod 城镇的酒馆。你应该会得到一个无法载入场景物品的错误信息（cannot load scene object）。我们还没有在这个场景中加入任何东西。哦，这里真暗……你会看到所有出现的人物都在同一个位置。这效果离正常效果相去甚远，所以我们需要编辑这一场景。在设置开启编辑模式的情况下，按Ctrl-E将开启编辑功能。现在你看出来为什么我们需要改为窗口模式运行了吧？这个工具栏将协助你完成你的工作。我不会详细介绍场景编辑模式的所有功能，因为我并不是这方面的专家。我会帮助你加入一些必须的东西，让你的酒馆看起来正常一点，并能为我们的任务服务。首先是一些基本控制操作：用基本方向键（A，W，D，S）来移动。按住鼠标左键并移动鼠标来旋转镜头。用鼠标右键点选场景中的物品。注意移动总是相对镜头朝向的，如果你对着地面按W 的话就会穿过地面。试着移动几下，找找感觉。我们的场景还需要一些物品使其正常运作，也可以增加一些东西来点缀场景。

需要加入的东西：

刷兵点（Entry points）

通道（Passage） (场景出口)

光源（增加亮度使得场景更容易看清）

我们会加入一些其他东西让场景显得不是那么空，但先让我们加入Entry Point。看看其他的酒馆场景，你需要给酒馆老板一个Entry Point，以及玩家进入场景时的Entry Point。

似乎酒馆老板的Entry Point 总是9，而玩家的总是0。为防万一，我们也照这样的方式做。在编辑模式工具栏上，点开“Add Object”（增加物品）按钮旁边的“item selection”（物品选择）下拉菜单，选择“Entry Point”。确认下方的窗口中“Entry Point”是被选中的。现在我们点一下“Add Object”按钮，会有一个黄色箭头随着鼠标移动，将其移动到地板正中并单击鼠标右键放置。如果你再移动鼠标，你会发现一个新的黄色箭头留在了地板上。再点一次”Add Object”按钮结束添加。

在“Add”（添加）功能栏上方的窗口你可以看到你已经添加了Entry Point 1。再上方一点是关于Entry Point的信息。如果我们右键单击游戏窗口中的Entry Point（在关掉“Add Obejct”功能后），可以选中它。按住G，它就会跟着鼠标移动。把它移动到吧台后面，你会发现它陷入了地板下面，我们需要把它抬起来一点。

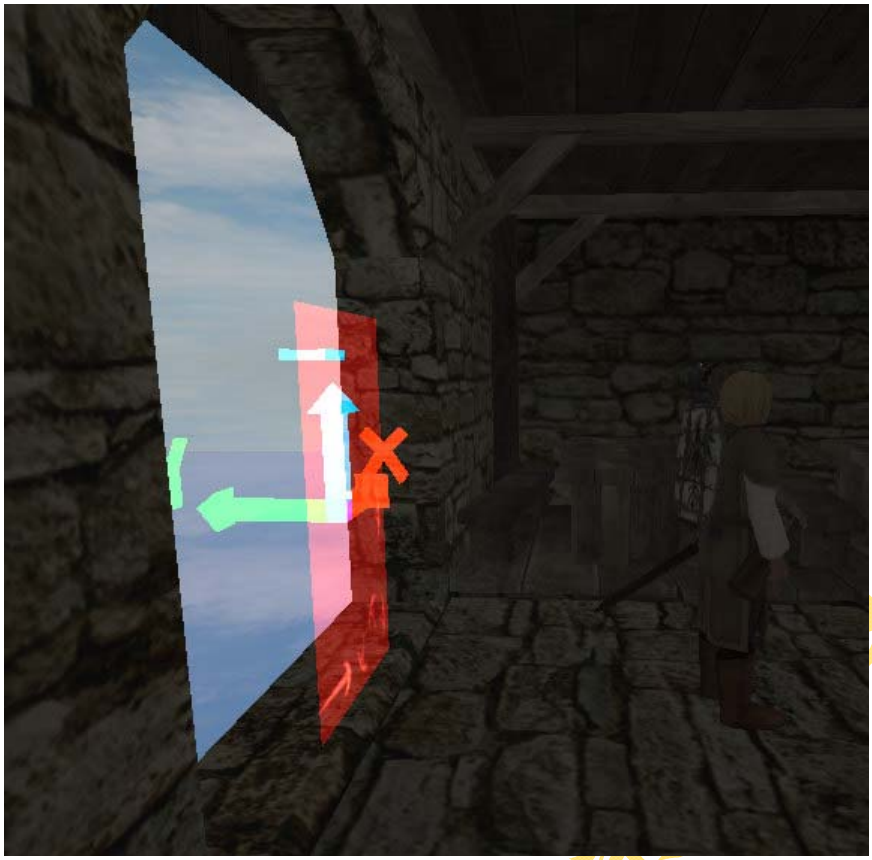
按住T 移动鼠标使其竖直移动。把它移上来直到可见。最后我们还需要酒馆老板面对外面的人群。按住Z来旋转它。这样你就放好了你的第一个场景物品了！在编辑模式的工具栏上点击右上方的“Help”（帮助）按钮你也能看到这些操作的说明。由于酒馆老板的Entry Point通常为9，我们要把这个Entry Point 的Entry No（刷兵点编号）改为9。你还不能立即看到变化，我们先保存一下。点“Close“（关闭）按钮，会弹出提示是否保存，选择“Yes”（保存）。离开酒馆并再次进入，打开编辑模式，你会在场景物品窗口看到Entry Point 9。在门口同样放上一个Entry Point 0 面向里面。

记住箭头指示了物品的摆放方式，Entry Point 0 是你进入房间的位置。

现在你应该在房间的各个地方多添加一些Entry Point。这些用于诸如雇佣兵，英雄，书商一类的visitor（来访者）。这些Entry Point 的编号必须在16 到24 之间。包括0 在内的这些Entry Point 都是内核处理的，Entry Point 9 不是，但这一个是原版的用法，所以我们也照搬。

再加入两个给我们的互动人物，最好放在屋子的两端。编号为5 和6.我把Entry Point 5 放在了门口（治安官），Entry Point 6 在上楼后的背后（Geoffrey）。这样在对话中我就会说Geoffrey在楼上。况且楼下已经很拥挤了！

四下看看你会发现一个空的大缺口。这里应该有个门，你也可以给它加上一个通向外面的Passage。尽管在本篇教程中不需要它也能完成我们的任务，但如果你的场景里有一扇可以通过的门会显得更真实，你也可能在其他任务中需要用到。Passage 可以让你无需返回菜单或地图而到达不同的场景。在“Item Selection”下拉菜单里面，选择Passage 并按下Add 按钮。如果你一时看不到它的样子，把你的鼠标移动到墙边，或者地板中央。它看起来比较奇怪，总之右键单机添加一个然后退出Add Object 模式。这样好点了。看起来像个红色的长方形。通过跟移动Entry Point 一样的方法把它移动到门口。不过要放在房间里。



（门内的Passage 标记）

在Passage 的数据窗口，我们将其Entry No（项目编号）改为4，menu item（菜单项）改为3。这是我在其他酒馆中看到的，所以就照搬吧。（译者注：Entry No 译者也不知道是做什么的，但这里的menu item 表示这个门通向的是在城镇菜单中第几个项目的场景，所以如果你更改了城镇菜单的顺序的话，所有依靠Passage 到达其他场景的门就会乱套）。

最后我们需要给这个空口安一扇门，让它看起来真实一点。选择Item type（物品类型）为Scene Props（场景物品），找到并选中“panel_door_b”。点击Add Object，把它加入场景中，退出Add Object 模式。我需要把它抬高一点点，然后只要放进门框就行了。

好了，最后我们要加入一些亮光。同样是在Scene Props 里面，有各种不同的光源。为了有些戏剧效果，把red light（红光）放在吧台后面的火炉中。然后在Scene Props 中找到fire_big 并添加到火炉里。如果你加上了光源又删除它，会发现它还在那里，但是你离开酒馆再回来看，就没了。你也应该注意到酒馆中的人就站在你分配的16-24 的Entry Point 上。可是酒馆老板在哪儿？由于每个酒馆老板是一个唯一的NPC，跟商人和马商一样，我们需要像新建其他人物一样新建一个酒馆老板。

你也可以根据喜好添加一些其他的Scene prop 进去。这是一个娱乐教程，所以尽情娱乐吧。做好之后保存退出。回到module system，打开module_troops.py。

8.3 主要人物

我们有了场景和人物，现在让我们把他们结合起来。打开module_troops.py，找到哈瑞克和Geoffrey 的位置。现在我们可以给他们指定所在位置了。我也给哈瑞克加上了新的脸型。做好后应该是这样：

```
# Add Extra Quest NPCs below this point
##JIK - new troop entry
["new_troop", "new_troop", "new_troops", tf_guarantee_boots | tf_guarantee_armor |
tf_guarantee_helmet, no_scene,reserved,fac_commoners,
[itm_sword_medieval_a, itm_fighting_axe, itm_leather_jerkin, itm_skullcap, itm_hide_boots],
str_9 | agi_9 | level(4), wp(80), knows_common, mercenary_face_1, mercenary_face_2],
["npc17","Geoffrey", "Geoffrey", tf_hero | tf_unmoveable_in_party_window,
scn_town_mod_tavern | entry(6), reserved,fac_geoffrey,
[itm_courtly_outfit, itm_hide_boots, itm_new_mace],
def_attrib | level(6), wp(60), knows_trade_3 | knows_inventory_management_2 | knows_riding_2,
0x000000019d004001570b893712c8d28d00000000001dc89900000000000000000],
["hareck", "Constable Hareck", "Constable Hareck", tf_hero, scn_town_mod_tavern | entry(5),
reserved, fac_commoners,
[itm_leather_jacket, itm_hide_boots],
def_attrib | level(5), wp(20), knows_common,
0x0000000e8000558036db6db6dbefb6db00000000001db6db0000000000000000],
```

最后，我们给吧台后面添加一个酒馆老板。我只是复制了另外某一个酒馆老板到酒馆老板列表的最末端，改动如下：

```
["town_mod_tavernkeeper", "Mod_Tavern_Keeper", "Mod_Tavern_Keeper", tf_hero |
tf_randomize_face | tf_female, scn_town_mod_tavern|entry(9), 0, fac_commoners,
[itm_woolen_dress, itm_nomad_boots],
```

def_attrib | level(2), wp(20), knows_common, woman_face_1, woman_face_2],

如果我们重新进入游戏，就能看到我们的人物和酒馆老板了。如果你跟他们对话，会进入默认对话，大致哈瑞克和Geoffrey 会是想要加入你的雇佣兵，因为我们还没有给他们创建任何对话。下一部分我们就将做这个对话。

第九部分

这一部分我们来学习module_dialogs.py，这是目前为止mod 系统中最大最重要的文件之一，包含了骑马与砍杀中所有的对话。此文件中包括了任何你想创建的新对话。要理解当每次一个对话开始时就从头到尾扫描这个文件，当其中某个元组符合条件时，执行该对话元组。

9.1 Module_Dialogs 解析

****注意：需要说明……是否以“anyone”开头的对话总是执行？我认为每次应该只有一个元组被选择执行……****

module_dialog.py 文件有点复杂。查看最前面的几个元组，你可以看到“anyone”对象，以及各种不同的对话开始声明（start-dialog state，稍后解释）。还包含有条件判断语句和寄存器（registrie），字符串寄存器（string registrie）以及$变量。看上去有点奇怪是因为这是每个人物都用到的判断对话文本，并不单独显示。这些对话用于让这些独个的信息正确出现在下面的对话文件中。通过这个办法，这些代码只要在这里输入一次，而不用在每个对白中重复输入。

如果对话的对象不符合某个元组的条件，就会被忽略。这个让你很容易地增加新的特殊NPC，村长，等等。

就像不用必须给马尼德设置专门的对话。

看看最上面的一个对话元组：

```
dialogs = [[anyone ,"start", [(store_conversation_troop, "$g_talk_troop"),
(store_conversation_agent, "$g_talk_agent"), (store_troop_faction, "$g_talk_troop_faction",
"$g_talk_troop"),
# (troop_get_slot, "$g_talk_troop_relation", "$g_talk_troop", slot_troop_player_relation),
(call_script, "script_troop_get_player_relation", "$g_talk_troop"),
(assign, "$g_talk_troop_relation", reg0),
(try_begin),
   (this_or_next|is_between, "$g_talk_troop", village_elders_begin, village_elders_end),
   (is_between, "$g_talk_troop", mayors_begin, mayors_end),
   (party_get_slot, "$g_talk_troop_relation", "$current_town",
   slot_center_player_relation),
(try_end),
```

下面还有很多，但是看到这里就可以了。这个用于判断当你开始和任何人（anyone）对话。看一下(try_begin)那块，我们看到它判断“$g_talk_troop”是否列在常量“village_elders_begin”到“village_enders_end”之间。因为使用了this_or_next is 的否定标识，这个条件和下一个条件成“或”关系。这个意味着如果第一个条件不符合，但是第二个(is_between, "$g_talk_troop", mayors_begin, mayors_end)符合，接下来的代码块还是会被执行。

否定标识和常量定义在module_constants.py。这就是为什么如果你要增加一个村长，他/她必

须要被放在"trp_village_1_elder"和"trp_merchants_end"之间，这样设置有利于让村长角色都能符合给他们的预定义设定。

让我们向下拉一点，找到一个比较特殊的对话，也就是和奴隶贩子拉蒙之间的对白：

[trp_ramun_the_slave_trader,"start",[(troop_slot_eq,"$g_talk_troop",lot_troop_met_previously,0),], "Good day to you, {young man/lassie}.", "ramun_introduce_1", []],

元组解析：

1) 对话对象。这要和与玩家对话的人物对应。通常是兵种ID（troop-id）。也可以是队伍模板ID（party-template-id），需要另外附加'|party_tpl' 代码（Hetairoi 注：例如："party_tpl|pt_bandits_awaiting_ransom"）。如果你希望这段对白适用于任何人的话就使用'anyone'常量。如果给这段加上'|plyr'意味着当前对话是由玩家所说。加上'|other(troop_id)'则意味着这段对话是由场景中的第三个人物所说。（前提是需要确认这个第三方存在于当前场景）
2) 对话起始(starting)声明。决定了台词的打开方式。对白的开始状态项必须吻合激活当前对话时的状态，使该段对白成为符合条件的可选项。如果该对白由在大地图上遇到另一支部队而激活，对话起始声明就以"start"开始（Hetairoi 注：应该是"party_encounter "之误）
   如果该对白由在城镇里面和另一个NPC 对话而激活，对话起始声明就以"start"开始
   如果该对白由帮助一支部队战胜别的部队而激活，对话起始声明就以"party_relieved" 开始
   如果该对白由解救一名俘虏而激活，对话起始声明就以"prisoner_liberated"开始
   如果该对白由战胜一支由英雄领导的部队而激活，对话起始声明就以 "enemy_defeated"开始
   如果该对白由触发器激活，对话起始声明就以"event_triggered"开始
3) 条件块(block)（列表）。必须是一个合法的操作块。参看header_operations.py。 = []
4) 对话文本（字符串）。
5) 对话结束(ending)声明。如果对话还有后续的下文，下文的对话开始声明就是前文的对话结束声明。
6) 结果块（列表）。玩家点击这个对话后执行的操作，必须为一个合法的操作块。参看header_operations.py。= []

观察拉蒙的开场白：

1) 对话对象 = trp_ramun_the_slave_trader
2) 对话起始声明 = "start"
3) 条件块 = (troop_slot_eq, "$g_talk_troop", slot_troop_met_previously, 0),
4) 对话文本 = "Good day to you, {young man/lassie}."
5) 对话结束声明 = "ramun_introduce_1"
6) 结果块 = []

对话声明，是这个元组中最需要说明的。现在我们将更深入地研究它们。

对话结束声明("ramun_introduce_1")把一个对话从一句台词引导到另一句。你可以任意指定对话结束声明，但必须有一个对话起始声明和它相对应。例如，如果我们想要写一个结束声明为"blue_oyster"的元组，这将会引导到起始声明为"blue_oyster"的元组。必须要有一个贴切

的配对，否则，build_module.bat 在生成时会抛出错误。

如果有多个元组的起始声明为"blue_oyster"，则会发生特别的事情。如果是玩家说的话，则会出现一个菜单供玩家选择；如果是NPC 说的，mod 系统会使用符合条件的第一个元组，而不管是否有多个元组符合条件。

要结束对话，必须使用"close_window"（关闭窗口）这个对话结束声明。

有数个特殊的对话起始声明供你选择，来启动一个对话。我们把这些称为对话初始(initial)声明。下面是对话初始声明的完整列表：

"start" —— 当在游戏场景中，和NPC 对话或者触发一个对话时使用。
"party_encounter" —— 当在大地图上遭遇另一个队伍时使用。
"party_relieved" —— 当玩家在大地图上援助一个战斗队伍并获胜后使用。
"prisoner_liberated" —— 当玩家战胜了拥有英雄俘虏的敌方队伍时使用。
"enemy_defeated" —— 当玩家战胜了由英雄领导的敌方队伍时使用。
"event_triggered" —— 当在游戏场景之外触发对话时使用。

正如你所看到的，每个对话初始声明都是为特定场合设计的，不会在其指定场合之外使用。我们的范例元组通过和Mod Town 的治安官哈瑞克（Constable）对话来启动，所以它的对话初始声明为"start"。

9.2 需求和条件块

对话接口的柔性很好，可以有许多种使用方法。它处理着大地图和游戏场景中的事件，让你能够随时触发对话。

接下来我们来讨论怎样最大限度地使用对话，然后学习如何自己创建复杂的对话。

上一节中我们提到，对话台词只有在所有条件都符合时才被使用到。首先，玩家必须和正确的部队说话，如果玩家和trp_ramun_the_slave_trader 对话，将不会调用trp_constable_hareck的台词。如果和玩家对话的所有人都会说某句台词，可以使用常量anyone。

其次，对话起始声明必须保持一致的场合——要么是由一个对话初始声明启动的，要么是从另一个台词的对话结束声明导入的。系统不会使用不符合规范的对话起始声明。

第三，条件块也遵循以上两条逻辑。系统只会使用满足所有条件的对话台词。只要条件块或对话起始状态有一个是错的，就会发生问题，有可能build_module.bat 抛出错误，也有可能游戏中的错误台词定死在那里，因为找不到对应的对话结束声明元组。

这就是你必须谨慎处理条件块的原因。确保你没有给你的对话设置错误的条件块，否则会使对话崩溃。

当然，如果一切都能协调工作，条件块将是非常强大的。条件块可以包含尝试(try)语句。你可以调用条件块内部的一个项，然后得到同一个条件块中的条件操作的结果，可以设置寄存器和字符串寄存器供实际对话使用。教程这一部分依次教你做这些工作。

你可以看到module_dialogs 的和奴隶贩子拉蒙的对话元组中关于条件块的使用。

[trp_ramun_the_slave_trader, "start", [(troop_slot_eq, "$g_talk_troop", slot_troop_met_previously,

0),],

这个块仅包含一个条件，即要求"$g_talk_troop"的slot_troop_met_previously 变量等于0，这是由于新游戏开始时所有寄存器和变量都将置为0。查看下一个元组，你将看到结果块：

[trp_ramun_the_slave_trader|plyr, "ramun_introduce_1", [], "Forgive me, you look like a trader, but I see none of your merchandise.", "ramun_introduce_2", [(troop_set_slot, "$g_talk_troop", slot_troop_met_previously, 1),]],

这一段中的troop_set_slot 操作将在台词显示后把"$g_talk_troop"（你的交谈对象）的slot_troop_met_previously 变量置为1。也就是说，这个台词一旦被显示，将永远不会再次出现——因为之后slot_troop_met_previously 变量将不再等于0。对话系统会忽略这句台词，跳到符合条件的另一个元组：

[trp_ramun_the_slave_trader,"start", [], "Hello, {playername}.", "ramun_talk",[]],

这个台词的唯一条件是玩家在场景中与Ramun 说话。Ramun 做完介绍后，每次同他说话都会调用这句台词。

值得注意的是这个文件最前面3 个元组里面的几个变量。用于你和……“anyone”对话时。每个都有其特殊的对话起始声明，但只用于你和某个特定的交谈对象之间的对话。其中的“$g_talk_troop”就是变量名单中的一个。在第一个元组的最初几行里，你和“$g_talk_troop”之间的关系值经过脚本troop_get_player_relation（参考module_scripts.py）计算后，从寄存器reg0 得到并保存到"$g_talk_troop_relation”：

```
[anyone ,"start", [(store_conversation_troop, "$g_talk_troop"),
(store_conversation_agent, "$g_talk_agent"),
(store_troop_faction, "$g_talk_troop_faction", "$g_talk_troop"),
# (troop_get_slot, "$g_talk_troop_relation", "$g_talk_troop", slot_troop_player_relation),
(call_script, "script_troop_get_player_relation", "$g_talk_troop"),
(assign, "$g_talk_troop_relation", reg0),
```

通过这种方法你就可以不用在每次添加新的对话的时候再做类似的事情。如果你想在某个特定对话中涉及一些相关数值比如你和谈话对象的关系值，它就可以帮助你扫描这些（或者别的特定的对话起始声明的）代码并且查看你需要的相关数值是不是已经存在。

9.3 添加新对话

我们终于有机会给我们的新兵种Geoffrey 设定独一无二的对话了。像module_troops 一样，你也不可以直接向module_dialogs 的列表底部添加一些元组。module_dialogs 的Python 列表末端，包括了针对所有人触发的对话占位符，由于文件是从头到尾扫描匹配台词的，所有在对话占位符之后添加的东东都将被忽略。

建议你在此注释的上方添加新对话：

COMPANIONS

我们的第一个目标是为Geoffrey 创建介绍语。在创建完整任务之前，我们将从这里获得一点经验。复制粘贴下面的元组到### COMPANIONS 注释之前：

JIKs added Dialogs
[trp_npc17,"start", [], "What? What do you want? Leave me be, peasant, I have no time for beggars.", "geoffrey_talk",[]],

首先我同样在前面加入了我的注释，在你写上大量的代码之前，最好先加上自己的注释。这是我们对话中的第一个元组。它由对话初始声明"start"启动，是Geoffrey 说的，引导到名为"geoffrey_talk"的对话起始声明。下面我们来创建一条紧跟着的台词，是玩家说的。复制粘贴下面的元组到刚才的元组下方：

[trp_npc17|plyr,"geoffrey_talk", [], "Nothing, never mind.", "close_window",[]],

这条台词是玩家说的，紧接着刚才第一条台词之后。由于对话结束声明为"close_window"，这条台词显示完毕后整个对话就会结束。

我们之前说过，添加多个具有相同对话起始声明的元组，游戏中将会显示一个选项菜单供玩家选择。记住，这个特性只对玩家的台词有效（例如[trp_npc17|plyr, ……），对其他兵种都无效（例如[trp_npc17, ……）。

复制粘贴以下元组：

[trp_npc17|plyr,"geoffrey_talk", [], "And who are you supposed to be?", "geoffrey_talk_2",[]],
[trp_npc17,"geoffrey_talk_2", [], "Why, I'm Geoffrey Eaglescourt, son of the Baron Eaglescourt! Leader of the Red Riders, bane of bandits,\
and crusher of pirates!", "geoffrey_talk_3",[]],
[trp_npc17|plyr,"geoffrey_talk_3", [], "Oh, I see. And how many pirates have you killed?", "geoffrey_talk_4",[]],
[trp_npc17,"geoffrey_talk_4", [], "See for yourself! I scalp every one of the dogs I kill. They are my battle trophies.", "geoffrey_talk_5",[]],
[trp_npc17|plyr,"geoffrey_talk_5", [], "That's nice. I'll be going now.", "close_window",[]],

这是一段Geoffrey 和玩家之间的完整的来回对话。目前为止没有重要事件发生，但是我们马上就要为之增加一点变化。为这段对话加入以下两个元组：

[trp_npc17|plyr,"geoffrey_talk", [], "Nothing, never mind.", "close_window",[]],
[trp_npc17|plyr,"geoffrey_talk", [(check_quest_active, "qst_mod_trouble"), (quest_slot_eq, "qst_mod_trouble", slot_quest_current_state, 0)], "You look familiar. Haven't I seen your face in a pigsty before?", "geoffrey_hostile",[]],
[trp_npc17|plyr,"geoffrey_talk", [], "And who are you supposed to be?", "geoffrey_talk_2",[]],
[trp_npc17,"geoffrey_talk_2", [], "Why, I'm Geoffrey Eaglescourt, son of the Baron Eaglescourt! Leader of the Red Riders, bane of bandits,\
and crusher of pirates!", "geoffrey_talk_3",[]],
[trp_npc17|plyr,"geoffrey_talk_3", [], "Oh, I see. And how many pirates have you killed?", "geoffrey_talk_4",[]],
[trp_npc17,"geoffrey_talk_4", [], "See for yourself! I scalp every one of the dogs I kill. They are my battle trophies.", "geoffrey_talk_5",[]],
[trp_npc17|plyr,"geoffrey_talk_5", [], "That's nice. I'll be going now.", "close_window",[]],
[trp_npc17|plyr,"geoffrey_talk_5", [(check_quest_active, "qst_mod_trouble"), (quest_slot_eq, "qst_mod_trouble", slot_quest_current_state,0)], "Really?\

Those scalps look suspiciously like horse tails to me.", "geoffrey_hostile",[]],

这两条台词会分别为起始声明为"geoffrey_talk"和"geoffrey_talk_5"的对话增加菜单项。技术上，具有相同对话起始声明的元组所在module_dialogs 中的位置是无关紧要的。对话系统会找到所有满足条件的元组并加到游戏菜单中。不过，出于代码直观性和可读性的考虑，你应当试着把相同菜单的元组放在一起。

这些台词中最显著的特点是条件块。只有在"qst_mod_trouble"任务被激活并且"slot_quest_current_state"等于0 时，这两条台词才会显示。现在复制粘贴最后的这些元组：

[trp_npc17,"geoffrey_hostile", [], "What?! I'll see you dead for that insult, peasant! Don't you know who I am?", "geoffrey_hostile_2",[]],
[trp_npc17|plyr,"geoffrey_hostile_2", [], "No . . . Was I supposed to remember?", "geoffrey_hostile_3",[]],
[trp_npc17,"geoffrey_hostile_3", [], "Why, I'm Geoffrey Eaglescourt, son of the Baron Eaglescourt, and you have delivered the gravest\
insult to my family's honour! I demand satisfaction! Meet me outside the town walls at noon, or you will be known a coward to every\
man in this countryside. Good day, {knave/wench}.", "geoffrey_hostile_4",[]],
[trp_npc17|plyr,"geoffrey_hostile_4", [], "Charming lad. Noon, eh? I shouldn't miss it...", "close_window", [(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,1)]],

mod 系统会根据玩家的性别显示文字，在“geoffrey_hostile_3”这句，根据玩家的性别显示'knave'或者'wench'。

这些做完后，我们现在就有了一段对话，它具备了任务相关的条件和结果，以及一个预先创建的任务（quest）元组。不过我们首先需要建立激活任务的一组条件。下一节中我们就来做这个。如果想，你可以运行build_module.bat 来访问Geoffrey，看看基本的交谈如何进行。

9.4 对话和任务

当添加新台词到已有对话中时，我们必须非常小心，保证新元组能和已有的协调好。记住，对话文件是从头至尾扫描的；记住检查你的条件块；记住仔细检查语法。一个拼写错误可以搞毁整块代码。经常编译生成你的mod，从而尽早捕获拼写错误。

首先我们要为Geoffrey 添加一个元组，加到他的其他台词之前。这意味着它会在其他台词之前使用到。如果这个元组满足条件，则只会使用这个元组，而不管其他台词是否也满足条件。

当前，Geoffrey 的第一个元组是：

[trp_npc17,"start", [], "What? What do you want? Leave me be, pesant, I have no time for beggars.", "geoffrey_talk",[]],

现在，在这个元组的上方复制粘贴下面的元组：

[trp_npc17,"start", [(quest_slot_eq,"qst_mod_trouble",slot_quest_current_state,1)],
"Noon time, {knave/wench}. I will deal with you then...", "close_window",[]],

放好这个元组后，Geoffrey 一旦向你发出决斗挑衅（"$geoffrey_duel"为1），则通常的对话将不再显示。通过这个途径，你就可以根据形势改变对话内容，创建包含条件块的变化多样的对话片段。由于老版本的M&B 的对话现在已经不能用，我们需要给你和治安官哈瑞克的交谈里添加一段简单的对话。

加在Geoffrey 的对话之后。这两句将很有用处：

[trp_hareck,"start", [], "Yes? Can I help you?", "hareck_talk",[]],
[trp_hareck|plyr,"hareck_talk", [], "Nothing. Good-bye.", "close_window",[]],

"hareck_talk"是一个多选项的对话菜单声明，刚才的元组是其中的最后一个选项。要在这个选项的上方显示另一条对话选项，必须在这个元组的上方添加一个新元组。

在刚才的两个元组的当中复制粘贴下面的元组，我们需要保留第一个Hareck 的“start”对话在上面：

[trp_hareck|plyr,"hareck_talk",[(neg|check_quest_active,"qst_mod_trouble"),(neg|check_quest_finished,"qst_mod_trouble")],
"Is something wrong? You look worried.", "hareck_troublemakers",[]],

只有"qst_mod_trouble"任务没有被激活和没有完成时，这个元组才会被显示出来。这是因为否定前缀 neg| 当条件不满足时使结果值为真。比如，eq 操作符要求两个值相等，neg|eq 要求值不相等。现在，在这个新元组的正下方复制粘贴下面的元组。我们要让“Nothing, Good-bye”元组在所有Hareck 的对话元组的最下面：

[trp_hareck,"hareck_troublemakers", [], "Oh, it's nothing, just . . .", "hareck_troublemakers_2",[]],
[trp_hareck|plyr,"hareck_troublemakers_2", [], "You can tell me, sir.", hareck_troublemakers_3",
[]],
[trp_hareck,"hareck_troublemakers_3", [], "No harm in it, I suppose. The trouble is, a few of the town's young nobles . . .\
Spoiled dandies and fops, the lot of them . . . they've decided that suddenly they're men to be respected, and that they\
should 'take matters into their own hands', to 'take action where the official government has failed'. They say they're going\
to kill all the river pirates that have been troubling Zendar of late. Of course, they've not actually gone out to fight any river\
pirates, but they've been making a great ruckus in town and there's not a thing I can do about it.",
"hareck_troublemakers_4",[]],
[trp_hareck|plyr,"hareck_troublemakers_4", [], "Hmm . . . Would there be a reward for solving this problem?", "hareck_troublemakers_5",[]],
[trp_hareck,"hareck_troublemakers_5", [], "What? What are you saying?",
"hareck_troublemakers_6", []],
[trp_hareck|plyr,"hareck_troublemakers_6", [], "Nothing, sir. However, it sounds to me like a neutral third party might be just\
what you need. I could talk to them.", "hareck_troublemakers_7",[]],
[trp_hareck,"hareck_troublemakers_7", [], "Heh. Well, you can try, friend. If you manage to do any good, I'll even throw in a\
few coins for getting the sand out of my breeches. Their leader is a boy named Geoffrey, spends

most of his time on\
watered-down ale and whores. Chances are you'll find him up the stairs in the back.",
"hareck_troublemakers_8",[]],
[trp_hareck|plyr,"hareck_troublemakers_8", [], "Thank you, constable. I shall return.",
"close_window",
[(setup_quest_text,"qst_mod_trouble"),(start_quest,"qst_mod_trouble"),
(quest_set_slot,"qst_mod_trouble",slot_quest_current_state, 0)]],
[trp_hareck|plyr,"hareck_talk",
[(check_quest_active,"qst_mod_trouble"),(quest_slot_eq,"qst_mod_trouble",slot_quest_current_state,3)],
"Constable, I've taken care of the toublemakers for you. They shouldn't be a worry any longer.",
"hareck_troublemakers_10",[]],
[trp_hareck,"hareck_troublemakers_10", [], "Truly? Thank God! A few more days and I would've thrown them all into a cell and thrown\
away the key. Here, take this. You've earned it.","hareck_troublemakers_11",
[(troop_add_gold,"trp_player",100),(add_xp_as_reward,750),(succeed_quest,"qst_mod_trouble")]],
[trp_hareck|plyr,"hareck_troublemakers_11", [], "My pleasure, constable. If you've any other jobs that need doing, please let me know. Farewell.", "close_window",[]],
[trp_hareck|plyr,"hareck_talk",
[(check_quest_active,"qst_mod_trouble"),(quest_slot_eq,"qst_mod_trouble",slot_quest_current_state,4)],
"Constable, I failed. I'm sorry.", "hareck_troublemakers_15",[]],
[trp_hareck,"hareck_troublemakers_15", [], "Oh . . . Oh well. I suppose you did the best you could. Thanks anyway, friend. Perhaps some other\
job will suit you better. I shall let you know when I have any. Farewell.", "close_window",
[(fail_quest,"qst_mod_trouble")]],

第一块代码设置了任务，详述了任务内容和如何开始这个任务。第二块代码以一些赏金和经验点数来结束这个任务，一旦任务失败"slot_quest_current_state"则设为3，我们要接着打败Geoffrey。第三块代码当玩家被Geoffrey 打败时结束这个任务（"slot_quest_current_state"设为4），意味着玩家将得不到任何奖励。否则我们就继续检测任务是否还是激活的。如果你不清楚变量这是个很好的回顾。在这里，即使任务结束，任务的slot_quest_current_state 仍然会是3 或者4，这意味着结尾对话仍然被使用因为条件仍然符合。我们需要增加的最后一样东西就是，我们要Geoffrey 在面对决斗的时候表现有所不一样。要达到这个，我们需要一个他在场并且准备好决斗的指示，我们设定当"slot_quest_current_state"为2 的时候就使用下面的对话，我们将把这个放在Geoffrey 的对话段的最前面。设好以后就像这样：

[trp_npc17,"start", [(quest_slot_eq,"qst_mod_trouble",slot_quest_current_state,2)],
"You have come to meet your doom, {knave/wench}...","geoffrey_duelling",[]],
[trp_npc17|plyr,"geoffrey_duelling", [], "Let's get this over with...",
"close_window",[(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,4)]],

我们还把"slot_quest_current_state"赋值为4。你也许还记得4 就是任务失败。这个保证了一旦你和Geoffrey开打，你要么赢（并且赢了就把状态设为3），要么输掉。正如你看到的，为了覆盖一个任务的方方面面，需要做不少事情。我们的任务只完成了一半，不过所有的对话

已经放置好了，现在准备进入教程下一部分，我们将掌握触发器和变量的改变。

第十部分

在我们继续之前，我们将快速结束我们的任务。如果我们威胁Geoffrey 离开城镇会怎样？这会使Hareck满意，从而完成任务。但是我们不能把它弄得太简单。我们可以做的一件事就是做一组对话，这组对话基于玩家的属性或技能，而他的属性或技能将影响他所说的话。通常强壮的大块头适合于威胁别人，所以让我们的快速任务基于玩家的力量值来完成。

[trp_npc17|plyr,"geoffrey_hostile_4",[(store_attribute_level,":player_str","trp_player",0), (ge, ":player_str", 14)],
"[Intimidate with Strength]","geoffrey_flee1",[]],
[trp_npc17|plyr,"geoffrey_hostile_4", [], "Charming lad. noon, eh? I shouldn't miss it...", close_window", [(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,1)]],

用附加的元组分派到对话的开始状态"geoffrey_hostile_4"。我们必须要做的第一件事是参照问题中的属性或技能。通过这么做，我们运行操作store_attribute_level，并且存储玩家的0 级属性（力量值）到一个局部变量“:player_str” 中。然后我们可以把它与我们设定的特定值相比较。14 是个很好的测试值，不过你也可以随意设置更大或更小的数。通过选择老兵、贵族的扈从、铁匠、个人复仇的背景，你可以很容易地达到14 或15 的力量初始值。你也可以打开作弊模式来升级。

最后，为了停止这个任务，你也可以把这个元组加到Geoffrey 对话的结尾：

[trp_npc17,"geoffrey_flee2",[],"Bah... This town is no fun anyway. Tell your constable that I will leave within the hour.","close_window",
[(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,3),
(remove_troop_from_site,"trp_npc17", "scn_town_mod_tavern")]],

这个将设置slot_quest_current_state 的值为3，这是和Hareck 谈话时所需要的成功状态。我们在这里所做的，是为自己提供了一种快捷的方式，来测试我们小任务的结束状态。我们也可以把Geoffrey 从场景中移除，尽管这不会见效，除非我们离开后再回来。作为对你所学的测试，你应该可以再在顶部添加一个与Geoffrey的“start” 对话，用来处理如果玩家离开这个场景之前想再次和Geoffrey 谈话的情况。确保测试针对任务的当前状态。Geoffrey 会说“我会在一小时内离开……”之类的话。

现在我们有了不少使用MOD 系统的经验，可以转入一些更加复杂的部分：触发器和脚本（Triggers 和Scripts）。触发器是基于时间的操作模块，可以每隔一段时间间隔或者在特定场合被激活。脚本在特定事件发生时运行，并且可以从其他模块文件中调用（使用call_script 操作）。当事件之一发生时，所有在操作模块中的操作都会被执行。

10.1 部队遭遇> Module_Scripts.py

一旦你的部队接触了另一支部队（定义在modul_parties.py），脚本“game_event_party_encounter” 就被执行。它是一个相当长的元组，但是通过使用“call_script”或者“jump_to_menu” 命令，它可被用来处理任何事件，并且其副本还设置

一些变量为其它元组所使用。

```
# script_game_event_party_encounter:
# This script is called from the game engine whenever player party encounters another party or a battle on the world map
# INPUT:
# param1: encountered_party
# param2: second encountered_party (if this was a battle
("game_event_party_encounter",
[
(store_script_param_1, "$g_encountered_party"),
(store_script_param_2, "$g_encountered_party_2"),# encountered_party2 is set when we come across a battle or siege, otherwise it's a negative value
# (store_encountered_party, "$g_encountered_party"),
# (store_encountered_party2,"$g_encountered_party_2"), # encountered_party2 is set when we come across a battle or siege, otherwise it's a minus value
(store_faction_of_party, "$g_encountered_party_faction","$g_encountered_party"),
(store_relation, "$g_encountered_party_relation", "$g_encountered_party_faction", "fac_player_faction"),
(party_get_slot, "$g_encountered_party_type", "$g_encountered_party", slot_party_type),
(party_get_template_id,"$g_encountered_party_template","$g_encountered_party"),
# (try_begin),
# (gt, "$g_encountered_party_2", 0),
# (store_faction_of_party, "$g_encountered_party_2_faction","$g_encountered_party_2"),
# (store_relation, "$g_encountered_party_2_relation", "$g_encountered_party_2_faction", "fac_player_faction"),
# (party_get_template_id,"$g_encountered_party_2_template","$g_encountered_party_2"),
# (else_try),
# (assign, "$g_encountered_party_2_faction",-1),
# (assign, "$g_encountered_party_2_relation", 0),
# (assign,"$g_encountered_party_2_template", -1),
# (try_end),
#NPC companion changes begin
(call_script, "script_party_count_fit_regulars", "p_main_party"),
(assign, "$playerparty_prebattle_regulars", reg0),
# (try_begin),
# (assign, "$player_party__regulars", 0),
# (call_script, "script_party_count_fit_regulars", "p_main_party"),
# (gt, reg0, 0),
# (assign, "$player_party_contains_regulars", 1),
# (try_end),
#NPC companion changes end
(assign, "$g_last_rest_center", -1),
(assign, "$talk_context", 0),
(assign,"$g_player_surrenders",0),
```

```
(assign,"$g_enemy_surrenders",0),
(assign, "$g_leave_encounter",0),
(assign, "$g_engaged_enemy", 0),
# (assign,"$waiting_for_arena_fight_result", 0),
# (assign,"$arena_bet_amount",0),
# (assign,"$g_player_raiding_village",0),
(try_begin),
   (neg|is_between, "$g_encountered_party", centers_begin, centers_end),
   (rest_for_hours, 0), #stop waiting
(try_end),
# (assign, "$g_permitted_to_center",0),
(assign, "$new_encounter", 1), #check this in the menu.
(try_begin),
   (lt, "$g_encountered_party_2",0), #Normal encounter. Not battle or siege.
   (try_begin),
      (party_slot_eq, "$g_encountered_party", slot_party_type, spt_town),
      (jump_to_menu, "mnu_castle_outside"),
   (else_try),
      (party_slot_eq, "$g_encountered_party", slot_party_type, spt_castle),
      (jump_to_menu, "mnu_castle_outside"),
   (else_try),
      (party_slot_eq, "$g_encountered_party", slot_party_type, spt_ship),
      (jump_to_menu, "mnu_ship_reembark"),
   (else_try),
      (party_slot_eq, "$g_encountered_party", slot_party_type, spt_village),
      (jump_to_menu, "mnu_village"),
   (else_try),
      (party_slot_eq, "$g_encountered_party", slot_party_type, spt_cattle_herd),
      (jump_to_menu, "mnu_cattle_herd"),
   (else_try),
      (is_between, "$g_encountered_party", training_grounds_begin, training_grounds_end),
      (jump_to_menu, "mnu_training_ground"),
   (else_try),
      (eq, "$g_encountered_party", "p_zendar"),
      (jump_to_menu, "mnu_zendar"),
   (else_try),
      (eq, "$g_encountered_party", "p_salt_mine"),
      (jump_to_menu, "mnu_salt_mine"),
   (else_try),
      (eq, "$g_encountered_party", "p_four_ways_inn"),
      (jump_to_menu, "mnu_four_ways_inn"),
   (else_try),
      (eq, "$g_encountered_party", "p_test_scene"),
      (jump_to_menu, "mnu_test_scene"),
```

```
    (else_try),
      (eq, "$g_encountered_party", "p_battlefields"),
      (jump_to_menu, "mnu_battlefields"),
    (else_try),
      (eq, "$g_encountered_party", "p_training_ground"),
      (jump_to_menu, "mnu_tutorial"),
    (else_try),
      (eq, "$g_encountered_party", "p_camp_bandits"),
      (jump_to_menu, "mnu_camp"),
    (else_try),
      (jump_to_menu, "mnu_simple_encounter"),
    (try_end),
(else_try), #Battle or siege
    (try_begin),
      (this_or_next|party_slot_eq, "$g_encountered_party", slot_party_type, spt_town),
      (party_slot_eq, "$g_encountered_party", slot_party_type, spt_castle),
      (try_begin),
        (eq, "$auto_enter_town", "$g_encountered_party"),
        (jump_to_menu, "mnu_town"),
      (else_try),
        (eq, "$auto_besiege_town", "$g_encountered_party"),
        (jump_to_menu, "mnu_besiegers_camp_with_allies"),
      (else_try),
        (jump_to_menu, "mnu_join_siege_outside"),
      (try_end),
    (else_try),
      (jump_to_menu, "mnu_pre_join"),
    (try_end),
(try_end),
(assign,"$auto_enter_town",0),
(assign,"$auto_besiege_town",0),
]),
```

这是列表中第一个也是唯一的触发器。他的结构很简单，只包含两个分开的域。

段的各个域剖析：

1）脚本（Script）ID：调用脚本时将在前面加上其前缀"script_"。
2）操作块：这必须是有效的操作块。参照header_operations.py。

通过加注释的描述，我们可以看出，当队伍在大地图上遇到了另一支队伍的时候，检测此触发器。我们现在将分别突出此段各个部分来检查它的作用。

第1 部分a：

```
(store_script_param_1, "$g_encountered_party"),
(store_script_param_2, "$g_encountered_party_2"),# encountered_party2 is set when we come
```

across a battle or siege, otherwise it's a negative

"store_script_param_#"是启动该元组的命令所传递的一些变量。正如注释显示的，第一个参数是你遇到的第一个部队，并且被存储到“$g_encountered_party”。 只有你遇到了一场已经在进行的战斗，第二个参数才会被设置，并且被存储到“$g_encountered_party_2”。 通过这些设置，我们可以使用这些变量命令一支部队进行任何操作。

第1 部分b：

```
(store_faction_of_party, "$g_encountered_party_faction","$g_encountered_party"),
(store_relation, "$g_encountered_party_relation", "$g_encountered_party_faction",
"fac_player_faction"),
(party_get_slot, "$g_encountered_party_type", "$g_encountered_party", slot_party_type),
(party_get_template_id,"$g_encountered_party_template","$g_encountered_party"),
```

正如我们在这里看到的，我们充分利用了新分配的变量。通过它，操作store_faction_of_party 可以被运行。通过这个操作，我们现在看到了相遇部队所属派别的信息，“$g_encountered_party_faction”,并且在那之后我们可以运行store_relation 并且把它和玩家的派别“fac_player_faction” 比较。这些操作可以在文件header_operations.py 里发现。

第2 部分：

```
#NPC companion changes begin
(call_script, "script_party_count_fit_regulars", "p_main_party"),
(assign, "$playerparty_prebattle_regulars", reg0),
```

在这个部分，我将进一步阐述。看那些变量（以“$”开头的），我们可以确定这个脚本是试图分配给适合战斗的NPC 的。向前稍跳一步，通过检查脚本“party_count_fit_regulars”我们可以看到这个是如何做到的：

```
#script_party_count_fit_regulars:
# Returns the number of unwounded regular companions in a party
# INPUT:
# param1: Party-id
("party_count_fit_regulars",
[
(store_script_param_1, ":party"), #Party_id
(party_get_num_companion_stacks, ":num_stacks",":party"),
(assign, reg0, 0),
(try_for_range, ":i_stack", 0, ":num_stacks"),
   (party_stack_get_troop_id, ":stack_troop",":party",":i_stack"),
   (neg|troop_is_hero, ":stack_troop"),
   (party_stack_get_size, ":stack_size",":party",":i_stack"),
   (party_stack_get_num_wounded, ":num_wounded",":party",":i_stack"),
   (val_sub, ":stack_size", ":num_wounded"),
   (val_add, reg0, ":stack_size"),
(try_end),
]),
```

一个短小的元组，内容很少并且容易明白。首先注意store_script_param_1 被存储到定义为“:party” 而不是“$party” 的变量中。这样做的原因是“:party” 只在这个元组里有用。这叫做局部变量。当元组开始时，它被创建，当元组结束时，它被删除。如果它被定义为全局变量“$party” ，它就可以在任何地方使用。除非你给它分配一个不同的值，否则它将保持原来的值。

接下来我们将简要讨论try_for_range 代码块。那些熟悉编程的人知道这将作为一个for…loop 循环。它和所有其它的lhs_operations 可以在header_operations.py 文件中找到，但是它们的作用不在这里定义。所以为了方便那些不熟悉循环的人，我将为你们把它分解开：

try_for_range = 调用循环
“i_stack” = 会增长的变量。如果你想要一个不断减少的计数，使用try_for_range_backwards
0 = i_stack 的初始值
“:num_stacks” = 当增长中的变量(:i_stack) 达到这个数，循环结束
try_end = 循环的代码块的结尾

在try_for_range 和try_end 之间的代码，将被重复执行直到循环结束。让我们回到主题。

第3 部分：

我将跳过分配代码，因为它在那只是再次设置变量，并回到第一个try_begin…try_end：

```
(try_begin),
  (neg|is_between, "$g_encountered_party", centers_begin, centers_end),
  (rest_for_hours, 0), #stop waiting
(try_end),
```

这一部分中有趣的内容是操作(try_begin) 。此操作打开一个尝试操作，它的功能和一般操作块相似，除了一点不同。如果一个尝试操作有一个条件没有满足，并不取消其他操作块全部；它只取消到最近的(try_end) 。所有(try_end) 之后的操作正常执行。事实上，一个尝试操作如同操作块中的一个独立操作块一样。你可以把它看成是一个if…end if 语句。

同样，代码中也存在另外的尝试操作(else_try) ，可以将其插入与一个主动尝试操作中，如同在module_simple_triggers 中的一样。一个else_try 块的内容只有在所有(else_try) 之前的尝试操作（包含之前的else_try 块）的条件都不能被满足的时候才有可能被执行，在下一节的后面一点的地方会看到。

第3 部分：

```
(try_begin),
  (lt, "$g_encountered_party_2",0), #Normal encounter. Not battle or siege.
  (try_begin),
    (party_slot_eq, "$g_encountered_party", slot_party_type, spt_town),
    (jump_to_menu, "mnu_castle_outside"),
  (else_try),
    (party_slot_eq, "$g_encountered_party", slot_party_type, spt_castle),
    (jump_to_menu, "mnu_castle_outside"),
  (else_try),
.
```

.
.

同样的，良好的注释让我们知道由于这里没有第二个部队，所以这不是你行进中的一次战斗或围攻。然后我们通过测试party_slot_eq，"$g_encountered_party"，slot_party_type，spt_town来了解我们遇到了哪种部队。如果"$g_encountered_party"不等于spt_town，那么我们转向下一个else_try。

旁注：如果我们搜索# Towns (loop) ，我们将会发现为每个城镇设置spt_town 值的循环。这里有更多的事情要做。比如在城镇里建立不同的场景，但是重要的是去标记（在第一行）它是否在towns_begin 和towns_end（都在module_constants.py 中定义）当中。这个代码块也向我们显示了我们为我们的城镇做的那些场景，必须与城镇的顺序相同。

这个将会继续，直到条件之一满足，然后在条件和下一个else_try（若它是最后的条件，则是try_end）之间的代码将被执行。随着这个的进行，你将会明白你可以有不同的非战斗遭遇。你也可以了解到这个检查可以用另外两种方法来做：

(is_between, "$g_encountered_party", training_grounds_begin, training_grounds_end),

用来测试module_constants.py 中定义的范围，以及：

(eq, "$g_encountered_party", "p_four_ways_inn"),

是定义在module_parties.py 中的对一个部队的直接参量。后者是个更特殊的定义，只在一些特殊情况下使用。如果你想使用城镇、城堡、村庄和其他多种部队的默认索引，那么你应该在module_constants.py 的定义范围内设置它们，并且按照处理其余目录那样处理它们。

第4 部分：

```
(else_try),
   (jump_to_menu, "mnu_simple_encounter"),
(try_end),
```

来到非战斗或非围攻的遭遇结尾，我们有“没有与任何条件符合”的最终结果。Jump_to_menu,

“mnu_simple_encounter”。 对于一长列复杂的try 块，适合用一个默认值来结束它。如果没有条件被满足，那么即使系统发出测试“错误”信息，你也可以确定发生了一些事情。我们用try_end 结束它。但是这不是总的结束。要记住，它由对第三支部队的检查而开始。现在我们已经分析完了“没有第二支部队”的结果。代码现在必须处理其他的结果，意味着你在行进中遇到了一场战斗或一次围攻。

第5 部分：

```
(else_try), #Battle or siege
   (try_begin),
      (this_or_next|party_slot_eq, "$g_encountered_party", slot_party_type, spt_town),
      (party_slot_eq, "$g_encountered_party", slot_party_type, spt_castle),
```

我先停在这里，因为我们有了另外一种形式的判断，这种判断可以拒绝一些条件。在try_begin声明之后，第一个条件以this_or_next|开始。虽然header_operations.py 没有为这个提供解释，但是我把它理解为：“如果这个条件为真，或者下个条件为真”。

第一个是判断是否party_slot_eq, "$g_encountered_party", slot_party_type, spt_town，第二个是判断是否party_slot_eq, "$g_encountered_party", slot_party_type, spt_castle。如果其中之一为真，那么它下面的代码块将会运行。要使下面的代码不运行，那必须使两个条件都为假。在其他的语言里，这个可以表达成：

If (<condition_1> or <condition_2>) then…

如果你对这个很熟悉并且要问“this_and_next 条件”在哪里，这很容易，你只要把一个条件放在另一个条件之后。当这些条件被计算时，如果有一个为假，那么try_begin 块就会停止。之前我们已经在对话里看到了这个，尽管当时我没有停下来去指出它。看这里：

[trp_hareck|plyr,"hareck_talk", [(check_quest_active,"qst_mod_trouble"),(eq,"$geoffrey_duel",3)], "Constable, I've taken care of the toublemakers for you. They shouldn't be a worry any longer.", "hareck_troublemakers_10",[]],

着重突出的条件必须都是真（check_quest_active 和$geoffrey_duel 要等于3）。成为IF (<condition1> AND <condition2>)的形式。

我们跳过下面的几个条件的检查，到最后两行：

(assign,"$auto_enter_town",0),
(assign,"$auto_besiege_town",0),

为那些在你的元组的过程中可能改变的变量，重新赋值是很重要的。你也需要清理那些可能要在其他代码块中使用的变量。如果初始值可能改变，那么当你下次在运行这个元组时，它的值可能不是原来的值了。

在一次或更多轮循环前，它可能保持一个值。$auto_enter_town 只在几行中用到，并且下次使用时，我们想要确保它有一个新值。我们要记住何时在native 代码里面使用$变量。

10.2 编辑部队相遇触发器 现在跳到9.3，需要说明

为使一个城镇有效，我们必须确保和这个城镇的任何相遇都把玩家指引向合适的游戏菜单。这个我们留下了两种选择：一种是我们指引我们的新部队使用Native 中的一个菜单，比如mnu_town；或者我们可以创建个新的菜单。

对于我们的new_town，我们将创建一个自定义菜单，然而如果你已经学过这些，那么你将知道使用默认菜单将更加容易。

对于我们想要做的，我们将把注意力放在第3 部分，分析spt_town：

```
(try_begin),
  (lt, "$g_encountered_party_2",0), #Normal encounter. Not battle or siege.
  (try_begin),
    (party_slot_eq, "$g_encountered_party", slot_party_type, spt_town),
    (jump_to_menu, "mnu_castle_outside"),
  (else_try),
    (party_slot_eq, "$g_encountered_party", slot_party_type, spt_castle),
    (jump_to_menu, "mnu_castle_outside"),
```

```
  (else_try),
```

高亮部分就是要分析的。如果我们想要为我们的城镇加一个新菜单，有2 个方法。如果只是一锤子买卖，那么我们可以添加另外一个try，用来检测你的城镇的特定ID。为了使事情变得有趣，让我们假设这是一个新的建筑，比如众多地牢中的一个。尽管我们也许想要保持城镇的其他方面（制作/定义场景）不变，我们还是想要一个特定的菜单来显示这个部队。我们可以到module_constants.py（在设置town，castle，和village 的附近），像这样自行设置：

```
towns_begin = "p_town_1"
castles_begin = "p_castle_1"
villages_begin = "p_village_1"
modtown_begin = "p_mod_town"
towns_end = castles_begin
castles_end = villages_begin
villages_end = "p_salt_mine"
modtown_end = castles_begin
```

现在我们有了一个仍在城镇范围内的范围，不过我们可以特别地指定p_mod_town，以及在它和p_castle_1之间的城镇。当像这样定义常量时需要仔细一点。你要确保你选用的名字尚未被使用。回到module_scripts.py，做出这些改变：

```
(try_begin),
  (lt, "$g_encountered_party_2",0), #Normal encounter. Not battle or siege.
  (try_begin),
    (party_slot_eq, "$g_encountered_party", slot_party_type, spt_town),
    (try_begin),
      (is_between, "$g_encountered_party", modtown_begin, modtown_end),
      (jump_to_menu,”mnu_modtown_menu”),
    (else_try),
      (jump_to_menu, "mnu_castle_outside"),
    (try_end),
    (party_slot_eq, "$g_encountered_party", slot_party_type, spt_castle),
    (jump_to_menu, "mnu_castle_outside"),
  (else_try),
```

高亮部分将在检查“$g_encountered_party”的slot_party_type 是spt_town 之后，检查“$g_encountered_party”是否是从midtown_begin 到midtown_end 的一部分。如果是，那么它将jump_to_menu（跳到菜单）“mnu_modtown_menu”。 如果不是，那么它将使用“mnu_castle_outside” 的默认菜单。如果我们去运行build_module.bat，那将会失败，因为我们还没有创建菜单“mnu_modtown_menu”。

现在为了使我们的城镇起作用，只剩下了一件事没有做，那就是为它创建一个菜单。在我们前往module_game_menus 之前，触发器中有更多的功能有待去探索。现在打开module_triggers.py，前往下一节。

10.3 Module_Triggers 解析

M&B 的MOD 系统中有两种触发器：简单触发器和扩展触发器。简单触发器在module_simple_triggers.py中，扩展触发器在module_triggers.py. 中。它们的工作原理相同，但扩展触发器有一些其他选项。让我们看看module_triggers.py。

```
# Tutorial:
(0.1, 0, ti_once, [(map_free,0)], [(dialog_box,"str_tutorial_map1")]),
```

这个文件中的第一个触发器很简单易懂。它是玩家第一次出现在地图上时弹出地图教程的触发器。

元组解析：

1) 检测间隔：检测触发器的间隔时间
2) 延迟间隔：在所有条件满足之后，确认触发器执行内容之前的时间长短。
3) 重新可用间隔：在执行一次触发器内容之后下次触发器变为可激活的时间。你可以把常量ti_once 放在这，以确保触发器激活一次后不会被再次激活。
4) 条件块（列表）：必须是有效操作段。参考header_operations.py。每次触发器被检查的时候，条件块将被执行。
若条件块返回真，则结果块将被执行。
如果条件块为空，则它一直被满足。即这个触发器一直触发。
5) 执行块（列表）（Consequences block 直译：结果块）：必须是有效操作段。参考header_operations.py。

指南元组检查：

1) 检测间隔 = 0.1
2) 延迟间隔 = 0
3) 重新可用间隔 = ti_once
4) 条件块 = (map_free,0)
5) 执行块 = (dialog_box,"str_tutorial_map1")

我们可以看出这个触发器每隔0.1 小时游戏时间被检测一次；没有延迟间隔；由于设置了ti_once 而不可重用。这意味着：如果所有此触发器的条件被满足，此触发器只执行一次，不再执行。因此，这个触发器在玩家被放到大地图任何地方的时候被触发。

现在我们可以设计自己的触发器了。拷贝教程触发器并将之粘贴在Python 列表最下端。

下面我们要对这个新触发器做的是在大地图上生成另一支队伍；叫做Geoffrey 的队伍。替换触发器条件块(map_free,0)如下：

```
(check_quest_active,"qst_mod_trouble"),
(quest_slot_eq,"qst_mod_trouble",slot_quest_current_state,1),
(store_time_of_day,reg(100)),
(gt,reg(100),12)
```

如果你记得我们在教程的第八部分中做的，当你同Geoffrey 的对话进行到决斗时，我们将变量”slot_quest_current_state” 赋值为1。因此，
(quest_slot_eq,"qst_mod_trouble",slot_quest_current_state,1) 检测Geoffrey 是否向你提出了挑战。如果没有，这个条件将不被满足。

接下来的操作是存储当前在一天中的时间到reg(100) 中，检测reg(100) 是否大于12；即一天中的时间是否在12:00 之后。如果是，条件满足，触发器被触发。

现在，替换执行块中的(dialog_box,"str_tutorial_map1")为下面的操作：

```
(set_spawn_radius,0),
(spawn_around_party,"p_mod_town","pt_new_template"),
(assign,"$geoffrey_party_id",reg(0)),
(party_set_flags,"$geoffrey_party_id",pf_default_behavior,0),
(party_set_ai_behavior, "$geoffrey_party_id", ai_bhvr_attack_party),
(party_set_ai_object,"$geoffrey_party_id","p_main_party"),
```

第一个操作设置了生成（部队的）半径；每次你要生成一个队伍的时候你都有做这个，否则队伍将会在最近一次设置的半径中生成，这是不可预知的。

下一个操作在("p_mod_town")——我们所创建的城镇，就在那里你遇见了Geoffrey——附近生成了一个队伍模版("pt_new_template")。由于这个操作在header_operations.py 中的定义方式，这项操作也将生成队伍的ID 存入reg(0)中。接下来我们将这个存入reg(0)中的ID 存入一个变量里，这样在reg(0)被改写的时候这个数据不会丢失。

最后两行，我们设置队伍的AI 行为和目标。这些操作很简单。我们叫"$geoffrey_party_id" 去攻击"p_main_party"。当你设置一个队伍攻击另一个队伍时，它将无情地追赶另一个队伍并且攻击它，不考虑所属势力或其他因素。

让我们再多加两行。完成后，你的Geoffrey 决斗元组看起来应像这样：

```
##JIKs Geoffrey duel quest trigger
(0.1, 0, ti_once,
[
(check_quest_active,"qst_mod_trouble"),
(quest_slot_eq,"qst_mod_trouble",slot_quest_current_state,1),
(store_time_of_day,reg(100)),
(gt,reg(100),12)
],
[
(set_spawn_radius,0),
(spawn_around_party,"p_mod_town","pt_new_template"),
(assign,"$geoffrey_party_id",reg(0)),
(party_set_flags,"$geoffrey_party_id",pf_default_behavior,0),
(party_set_ai_behavior, "$geoffrey_party_id", ai_bhvr_attack_party),
(party_set_ai_object,"$geoffrey_party_id","p_main_party"),
(remove_troop_from_site,"trp_npc17","scn_town_mod_tavern"), #Since he is in the field, take him from the tavern
(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,2) #Allow of a new conversation to start the duel
]
),
```

最后两行为进入决斗做了一点清理。我做的注释可以向自己解释代码，你写代码做注释时也应该这样。现在当中午（12 点）来临，部队“new_template”（Geoffrey 所在的部队）产生时，在Mod 城镇的旅馆里不再能够找到他。我们也已经把变量”slot_quest_current_state”提升到2。

保存你的进展，运行build_module.bat。那么现在我们的任务已经准备开始了。如果你现在就开始这个任务，你会发现无论与Geoffrey 及他的打手们的战斗结果如何，你都会使任务失败。这是因为在这里，我们还没有把“slot_quest_current_state”改到3 的代码，即任务成功的状态。

****到这里为止都是有效的教程****

需要制作当你对付GEOFFREY 和他的爪牙时的对话。需要考虑当你战胜其他领主时的对话。

第十一部分

接下来我们将关注的是任务模板模块（module_mission_templates.py），这个文件包含了所有战斗的代码，包括野战，攻城和决斗，以及一些特殊战斗，比如刺杀商人之类的。如果你需要调整战斗的话，这就是你需要编辑的文件。谢谢 Phosphoer 和 MartinF 的协助，我们先来试试做一个决斗的剧本（在竞技场中 1对 1）。之后我们来看看并修正野战的场景。

11.1 Module_Mission_Templates 解析

每次当你面对战斗的时候，这些代码都会被检查一遍，你也可以设置一些快捷键（比如 TAB）并在战斗中使用。在载入任务模板（mission_templates）之前，系统会设定一些常量。前两个用来设定装备的更换的，我们以后再来研究它，如果你正在研究编码的话你应该知道这些。接下来是一些其它的长的元组内容，比如 common_battle_mission_start。你应该明白这些数值的格式是和 module_triggers.py 类似的，因为它们都是触发器。你可以发现 module_mission_templates.py 文件头的第六段的代码就是触发器。通过在文件头把一些常用的触发器定义成常量，你就不需要给每个需要的元组都输入这些触发码代码了。

你可以在第 650 行左右找到 mission_templates(mission_templates = [ )的开头。我们来分析定义在文件头的

第一段元组“town_default”：

1. 任务模板 ID（前缀 mt_） = "town_default"
2. 标识（定义在 header_mission_template.py） = 0
3. 任务类型整数。必须是’charge’或者’charge_with_ally’，再或者必须为-1 = -1
4. 任务描述文本 = "Default town visit"
5. 刷新单位记录列表（列表）。详细列表（刷新单位的入口点号，刷出的新单位对应的标识，换装/装备覆盖标识，AI 标识，刷出的兵种代码，给刷出的兵种的额外装备——最多 8 样） =

```
[(0,mtef_scene_source|mtef_team_0,af_override_horse,0,1,pilgrim_disguise),
(1,mtef_scene_source|mtef_team_0,af_override_horse,0,1,[]),
(2,mtef_scene_source|mtef_team_0,af_override_horse,0,1,[]),
(3,mtef_scene_source|mtef_team_0,af_override_horse,0,1,[]),
(4,mtef_scene_source|mtef_team_0,af_override_horse,0,1,[]),
(5,mtef_scene_source|mtef_team_0,af_override_horse,0,1,[]),
(6,mtef_scene_source|mtef_team_0,af_override_horse,0,1,[]),
(7,mtef_scene_source|mtef_team_0,af_override_horse,0,1,[]),
(8,mtef_scene_source,af_override_horse,0,1,[]),
(9,mtef_scene_source,af_override_horse,0,1,[]),
(10,mtef_scene_source,af_override_horse,0, 1,[]),
(11,mtef_scene_source,af_override_horse,0,1,[]),
(12,mtef_scene_source,af_override_horse,0,1,[]),
(13,mtef_scene_source,0,0,1,[]),
```

```
(14,mtef_scene_source,0,0,1,[]),
(15,mtef_scene_source,0, 0,1,[]),
(16,mtef_visitor_source,af_override_horse,0,1,[]),
(17,mtef_visitor_source,af_override_horse,0,1,[]),
(18,mtef_visitor_source,af_override_horse,0,1,[]),
(19,mtef_visitor_source,af_override_horse,0,1,[]),
(20,mtef_visitor_source,af_override_horse,0,1,[]),
(21,mtef_visitor_source,af_override_horse,0,1,[]),
(22,mtef_visitor_source,af_override_horse,0,1,[]),
(23,mtef_visitor_source,af_override_horse,0,1,[]),
(24,mtef_visitor_source,af _override_horse,0,1,[]),
(25,mtef_visitor_source,af_override_horse,0,1,[]),
(26,mtef_visitor_source,af_override_horse,0,1,[]),
(27,mtef_visitor_source,af_override_horse,0,1,[]),
(28,mtef_visitor_source,af_override_horse,0,1,[]),
(29,mtef_visitor_source,af_override_horse,0,1,[]),
(30,mtef_visitor_source,af_override_horse,0,1,[]),
(31,mtef_visitor_source,af_override_horse,0,1,[]),],
```

6. 触发器列表（参考 module_triggers.py 文件的触发器格式） =

```
[
(1, 0, ti_once, [], [
      (store_current_scene, ":cur_scene"),
      (scene_set_slot, ":cur_scene", slot_scene_visited, 1),
      (try_begin),
        (eq, "$sneaked_into_town", 1),
        (call_script, "script_music_set_situation_with_culture", mtf_sit_town_infiltrate),
      (else_try),
        (eq, "$talk_context", tc_tavern_talk),
        (call_script, "script_music_set_situation_with_culture", mtf_sit_tavern),
      (else_try),
        (call_script, "script_music_set_situation_with_culture", mtf_sit_travel),
      (try_end),
  ]),
(ti_before_mission_start, 0, 0, [], [(call_script, "script_change_banners_and_chest")]),
(ti_inventory_key_pressed, 0, 0, [(set_trigger_result,1)], []),
(ti_tab_pressed, 0, 0, [(set_trigger_result,1)], []),
],
),
```

我们将做一个和原版为女士的荣誉决斗任务类似的新的任务模板。我们把新的加在文档的末尾，在最后一个 ‘]’ 之前。

现在你应该根据顶端的文件信息核查过元组的格式了，所以我将忽略这一部分，而开始做我们的那个决斗的场景。向下一直到文档的末尾。虽然叫做“任务模板”，其实顺序倒是无所谓。找到文档的结尾处在最后那个‘]’之前留出一点空间。研究一下这儿的那些决斗代码，我

们可以自己完美地做一个和 Geoffrey决斗的场景。

和所有的元组一样，我们的开始处应该是圆括号 ‘(‘。在代码的结尾处的圆括号后应该是逗号 ‘),’,写入 ID，看上去应该像这样：

(“arena_duel_geoffrey”,

),

根据元组的描述我们知道还需要写入任务的标识和任务的类型。决斗将在竞技场内举行，所以写上mtf_arena_fight 标识，类型为 -1。另外我们知道决斗既不是 charge 也不是 charge_with_ally，所以应该为-1。接下来是任务的文字描述，就写入 “Dueling Geoffrey” 吧。现在看起来应该是这样：

(“arena_duel_geoffrey”,mft_arena_fight,-1,
“Dueling Geoffrey”,

),

注意用逗号把每段代码隔开，我们现在来研究更复杂的元组。首先设定任务中不同的队伍吧，刷人记录的元组和任务模板中的元组类似，第一个是队伍的入口点。有些特殊场景的入口点很重要。我们的场景将使用一个标准的竞技场。搜索 “arena_melee_fight”，你可以找到刷人记录代码的例子。我们要找的是下一个元组“arena_challenge_fight”就在我们现在修改的地方附近。为我们的两位决斗者做两段入口点的代码，就用出生点 56 和 58。如果你去编辑模式看看的话可以发现这 2 个点都在竞技场中央部分。接下去就按照我们看到的去做。我们将来还会经常在这个决斗任务模板里面以这两个元组为例子。

既然我们的决斗和 arena_challenge_fight 的代码都差不多，就复制这两段刷人记录来作为我们的元组，我们来看看刷人记录的剩余列表：

(56, mtef_visitor_source|mtef_team_0, 0, aif_start_alarmed, 1, []),
(58, mtef_visitor_source|mtef_team_2, 0, aif_start_alarmed, 1, []),

设置刷人入口点之后我们还需要设定刷人的标识，第一个标识设定冲突模式，这里我们使用 mtef_visitor_source，这个也可以设置为 mtef_attackers 或者 mtef_defenders，可是它们是用在野战中的。

攻击方是引起遭遇的一方（一般都是玩家啦），另一方就是防御方了。

第二个标识(位于‘|’之后)mtef_team_# 设定刷出的兵种加入哪方，范围从 mtef_team_0 到mtef_team_3，最多能有 4 支队伍。现在看看“换装”标识，它是用来设定哪些物品（武器衣甲之类）不能在特定的场合出现（装备）。我们的决斗是步战，需要把马匹给禁了。下一个是 AI 标识，一般都设定为aif_start_alarmed，这里我们也这么做。这让 AI 准备战斗。这些都完成后我们要设定每方刷出的人数。既然是单挑就设定为刷出一人。

最后一串位于‘[]’之中的列表设定了刷出来的人的装备，一人最多 8 件。如果你把他们自身的所有的装备都禁掉的话，士兵就会从这个列表里面拿取他们需要的装备。如果你想让这里刷出来的士兵在木棍和木剑之中选择一样，禁掉所有的装备（使用换装标识 af_voerride_all）然后添加供随机选择的装备。

现在你的 mission_template 应该看起来像这样：

```
(“arena_duel_geoffrey”,mft_arena_fight,-1,
“Dueling Geoffrey”,
(56, mtef_visitor_source|mtef_team_0, af_override_horse, aif_start_alarmed, 1, []),
(58, mtef_visitor_source|mtef_team_2, af_override_horse, aif_start_alarmed, 1, []),

),
```

我们现在有 2 个在不同地点刷出来的不让骑马的人，这样就差不多了。现在需要设置一些触发器，每隔一段时间来检查某些事情有没有发生，比如某人被打晕，战斗的时间限制或者开门之类的场景的转换。现在，我们来试试某人被打晕。我们有 2 个决斗者，自然需要 2 个触发器，每人一个。

我们还得不让玩家在决斗时打开物品栏，这儿有一个预处理触发器叫做 common_inventory_not_available. 这将是一个固定触发器。

先来假设是你被打晕了。我们得设定合理的频率来检测这个，可以设为在你倒地几秒之后。记住触发器元组的格式，首先得研究一下什么时候激活触发器。我假设在开始战斗几秒内，接下是检查的频率，3 秒一次就差不多了。我们还得确定触发器不会被反复激活。幸运的是有一个检查你是否被击倒的语句(main_hero_fallen)。有了这个我们还可以设置触发器失效的时机，还有任务失败的提示并退出任务。你的任务模板现在应该像这样了：

```
(“arena_duel_geoffrey”,mft_arena_fight,-1,
“Dueling Geoffrey”,
(56, mtef_visitor_source|mtef_team_0, af_override_horse, aif_start_alarmed, 1, []),
(58, mtef_visitor_source|mtef_team_2, af_override_horse, aif_start_alarmed, 1, []),
   [common_inventory_not_available,
      (ti_tab_pressed, 0, 0, [(display_message, "@Geoffrey: Trying to flee, Peasant?")], [])
      (1, 3, ti_once,[(main_hero_fallen),],
         [(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,4),
         (finish_mission),
         ]
      ),
   ],
),
```

我们用预定义代码 finish_mission 来结束这个任务模板。如果不这样的话你会一直晕在地上，除非你退出游戏，玩家或许也可以用 TAB 来退出决斗，不过我们不想那样，因为那也许会让这场决斗中途流产。在 common_inventory_not_available 之后我们来加上一个基于按下 TAB 键的触发提示。就是：

```
(ti_tab_pressed, 0, 0, [(display_message, "@Geoffrey: Trying to flee, Peasant?")], [])
```

ti_tab_pressed 是一个某个触发器，发生在当我们按 TAB 键的时候，这样的话会显示 Geoffrey 的话(Geoffrey 说：想逃跑?你这个乡巴佬！)。我们要让玩家坚持到决斗结束。

最后来看看玩家胜利的情况，和打输的差不多，我们可以用这条代码：all_enemies_defeated，来检测玩家是否打倒了所有的敌人。如果玩家打赢那么这个值就设

为 1.还有一种情况是双方同时被打晕了（比如被弓弩射晕），如果这样的话就算玩家输了。在结尾处别忘了改动 slot_quest_current_state 来结束 missiontemplate。完成的看上去差不多应该是这样：

```
(“arena_duel_geoffrey”,mft_arena_fight,-1,
“Dueling Geoffrey”,
(56, mtef_visitor_source|mtef_team_0, af_override_horse, aif_start_alarmed, 1, []),
(58, mtef_visitor_source|mtef_team_2, af_override_horse, aif_start_alarmed, 1, []),
   [common_inventory_not_available,
   (ti_tab_pressed, 0, 0, [(display_message, "@Geoffrey: Trying to flee, Peasant?")], [])
   (1, 3, ti_once,[(main_hero_fallen),],
      [(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,4),
      (finish_mission),
      ]
   )
   (2, 3, ti_once,[( all_enemies_defeated, 1),(neg|main_hero_fallen, 0),],
      [(quest_set_slot,"qst_mod_trouble",slot_quest_current_state,3),
      (remove_troop_from_site,"trp_npc17","scn_town_mod_tavern"), ##Remove Geoffrey from
      the tavern scene
      (finish_mission),
      ]
   ),
   ],
),
```

我又加入了一点东西来修正这个，如果我不把 Geoffrey 从场景中移去的话他会不合时宜的说话，你应该注意到了如果他赢的话我就不必把他移除了，我故意这样的。你也可以试试设定如果他打输了也会说一些话（以 slot_quest_current_state 的值为基础），我会在下一部分中把这些都调整好。

第十二部分

有纹章的甲胄上的纹章一般都会是家徽/旗帜/家纹上的团，但并不仅限于这样，我会向你展示如何让你的士兵们穿成你队伍的颜色。如果你已经整理过你的旗帜（比如我有8 个不同颜色的旗帜），你可以让你的背景颜色和你的旗帜颜色相对应。用以下指令:

(troop_set_slot, "trp_banner_background_color_array", 0, 0xFF8f4531),

你可以在 module_scripts.py 的第一段文本中找到那条显示旗帜的指令。

12.1 贴图和Alpha通道

先从这个基本的衬衫来开始（foxyman注：这个衬衫模型在/CommonRes/item_meshes1.brf 里，就叫shirt，可找死我了……），我将会把它弄成和旗帜背景一样的颜色。注意已经通过代码将背景色与旗帜颜色设为一致了。

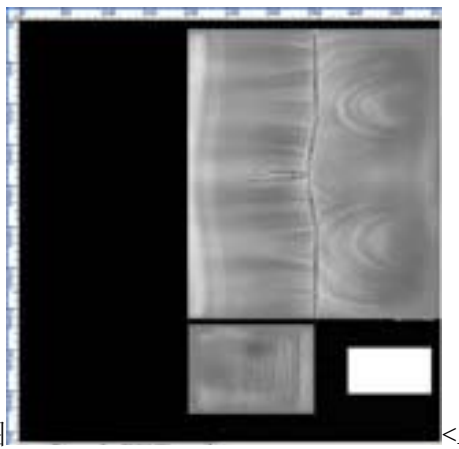
这可以在module_scripts 中第一个script 的开始处设置。首先，让我们提取他的纹理并加上新的alpha 通道。

打开 costume1.dds（/Textures/costume1.dds）文件。新建一个dds，把贴图大小设定成 512x512，从costume1.dds 复制衬衫的贴图（中间偏上的位置）以及其相同颜色的袖筒，还有棕色的裤子（其实随便哪个都可以）到你的新文件。

我把衬衫的贴图挪到一边，让旗帜有更多的空间，我习惯把旗帜放在左侧。通过测试我总结出，按我的放法，旗帜总是在从 0，0（左上） 到 195， 512 这个区域；于是将这区域标注（用另一个图层）， 在绘制其他部分（衬衫， 袖筒，裤子）时空出这片区域。 如果你像其他的tableau_materials（可变材质）一样摆放的话，旗帜则在贴图的中央。

因为我改过这个贴图和它的UV 展开图，所以我了解贴图模型的对应关系。 你应该自己先用原展开图试验一下。 根据SantasHelper 的指导，我们在原帖图上想要显示旗帜颜色的部分添加alpha 通道，使其覆盖衣服贴图。 裤子我就不做了，它仍然保留本来的颜色。选取衬衫部分的贴图，复制到 alpha 上，我应用了反色效果，并在亮度和对比度上稍稍修改了一下， 使用黑色背景 (通用效果)，效果如下：

<贴图


<Alpha

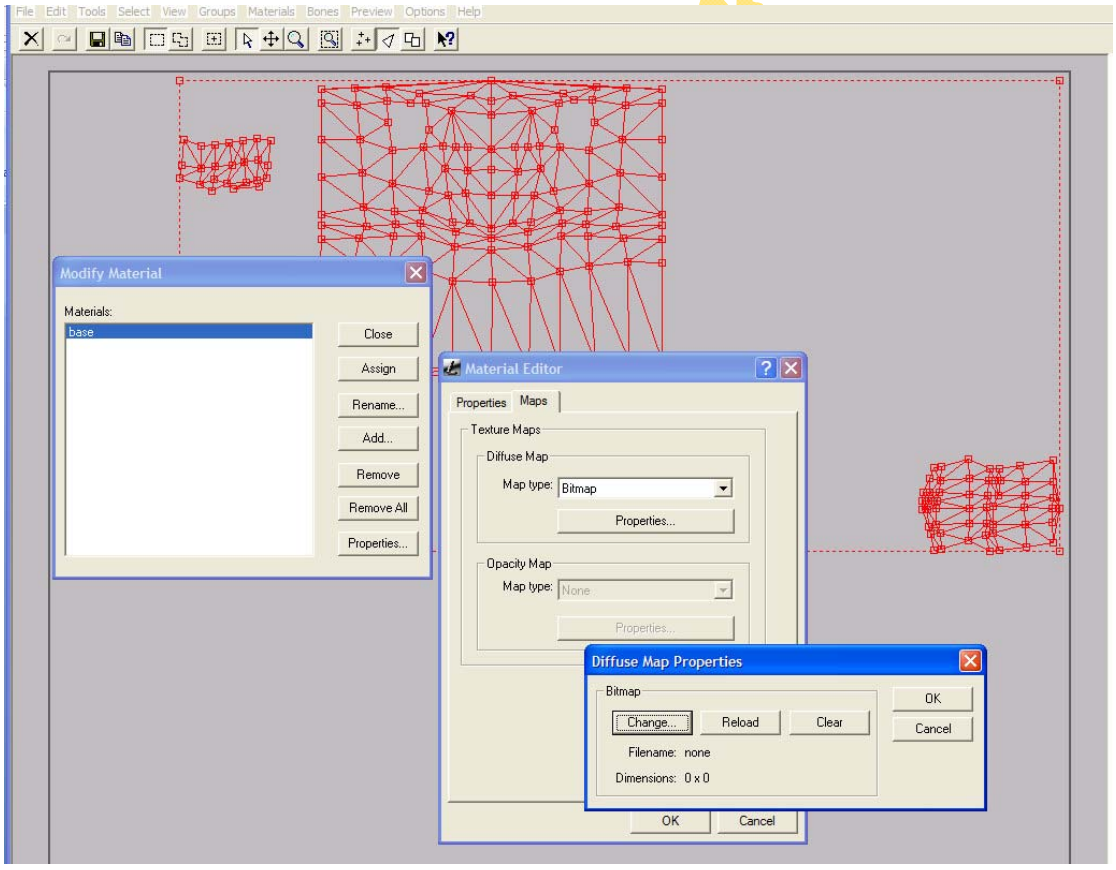
将这个保存为 j_cloth01.dds， 一定要保存为 DXT5 格式。 贴图完成后还需要展开到网格模型上。 我用的是 BRFeditor 9.8.5 和一个叫做LithUnwrap 的免费 UV 展开软件， 还有非官方的MB 工具obj2smd.exe。 你可以在这个链接下载 obj2smd.exe。
http://forums.taleworlds.net/index.php/topic,55698.0.html

(谢谢Manitas 提供).

12.2 UV 展开

用 BRF editor 把这衬衫导出成两种格式，OBJ 和 SMD。导出成 SMD 的时候别忘了选择骨骼。用LithUnwrap 载入那个 OBJ 文件（File>Model>Open)，你会看到窗口显示出UV 展开的线框。我们现在需要把我们的材质加上去。点击 Material>Modify… ， 如错你看到有任何素材的话，点选它们并删除。

点击“Add”，随便起个名字，我的叫做 'base'。选中，点击属性(Properties)。在属性窗口中点击 ”Maps”标签。将Diffuse map 栏改为 bitmap 格式， 再点击属性。点击“Change”，选取你刚刚做的 贴图（我的是 j_cloth01.dds）。



选取之后，点击 OK 两次。 你应该已经返回到 Modify Material 窗口。 确认你的材质还是在选区中，点击 Assign。只有你选取中模型面这才有效， 否则 Assign 键是点不了的。这可以让我们在打开预览的同时看到网眼上的素材。点击 Close。点击 View>Material， 选取你的材质并应用，你可以在背景中看到它。

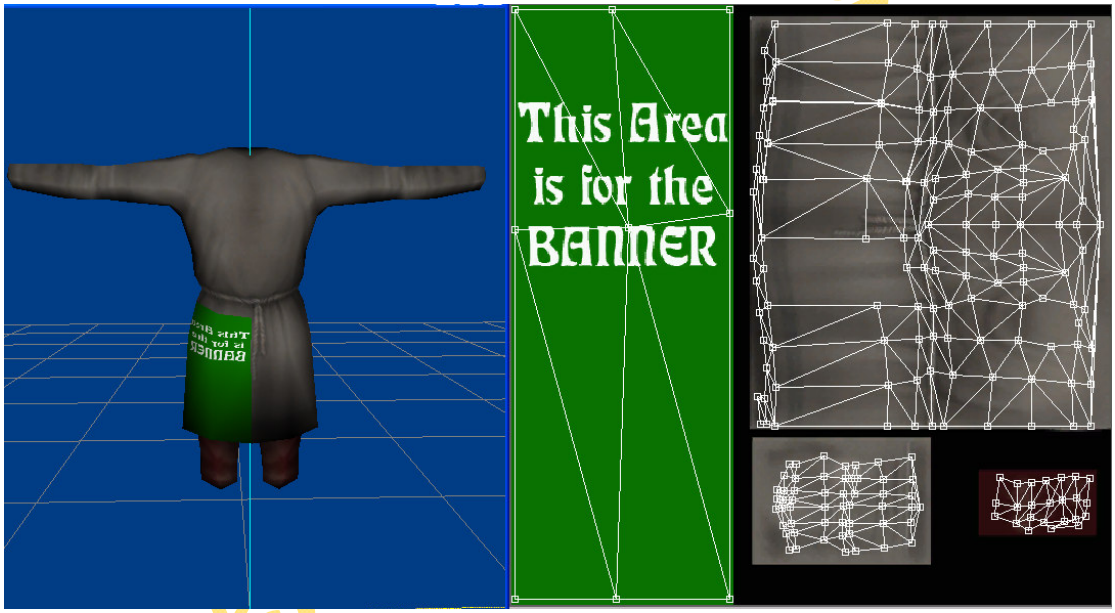
我们需要将UV 展开调整到与贴图贴合一致。

我先把袖筒和裤子挪到正确的位置，再旋转使衬衫的主体和贴图的方向相对应（逆时针旋转90 度）。尺寸有点儿不吻合，因为初始的尺寸是 1024x1024， 我们把它转成了 512x512，现在将它放大2 倍就好了。

（在 Edit 菜单中修改）。 我不会十分详细的解释这些软件的操作，你得自己去摸索和练习。 我只告诉你现在点击Preview>Show Model，你能看到贴图的效果。 现在我把它们都调整好了，这个贴图的一个好处是腰带绳位于正中央，我们可以用它来区分正面与背面，也可以用来定位纹章。把它放在正中央，左胸还是背后取决于你，我将用腰带下方一侧的那块地方来展示。

这个软件的一个缺点是面的选取，要选取一面，你得从顶点罩住它，所以你会把其他多余的面一起选进来。

和其他软件不同，放弃选取是按住 Ctrl 而不是 Alt，用它来去除不需要的面。我把顶点向左平移了一点再从右侧来选取，再把它们挪到空处。我把它放大了一点儿（大约2.5x），现在看上去效果如下：



左侧是预览，右侧是 UV map

如你所见预览中的黑色区域是我放入的 UV map 的那部分。 当定义 tableau_material 的时候，这是纹章将会出现的地方，面积也和那个差不多。 这儿已经完成了，点击 File>Model>Save 来保存。确认把它储存到正确的文件夹下（我经常错把它存到 texture 文件夹下，因为那是我最近储存过的文件夹），别忘了把它存为 OBJ 格式。

储存时会有一个相关的窗口弹出，全打上勾。

接下来我们要把那个 OBJ 和 SMD 给合并来完成衣服。很不幸的是，导出的 OBJ 不会储存动画信息，SMD 倒是可以。Manitas 的那个 EXE 文件会合并它们。如果你有 UVmapping 软件可以编辑 SMD 文件的话，你可以跳过这部，直接合并它们。

在 obj2smd.exe 文件夹下中有一个 readme，我就不多说了。你结束之后把 in.obj 和 out.smd 改成你想要的名字，我的是 j_cloth01.obj 和 j_cloth01.smd。最好把他们都存下来，以后也许会有需要改进的地方。 那个 in.smd 已经没用了，你可以导入新的 SMD 文_____件到

BRF editor 中。 别忘了看看 readme 中的相关介绍与提示。

12.3 BRF 制作

我不是来教怎么用 BRF editor 的，基本的你们都应该会，否则不该来学习这章。

打开 BRF editor，在导入模型之前先做一点设置。首先需要在BRF editor 中加入那个贴图。点击 Tex 栏，点击下方的”add"，会出现一个默认的 texture DDS 文件名，把它改成你的DDS 的名字，我的是j_cloth01.dds。之后点击 Mat 栏，我会用比较耗时的方法因为我的多选框有些问题。点击导入，在CommonRes 文件夹（位于骑砍中文件夹中，不在MOD 中）中找到heraldic_armors.brf。现在导入2 件 材质，heraldic_armor_a 和 sample_heraldic_armor_a。

当它们在你的 materials 区域后， 我们将它们改名使其符合命名格式， 我的是j_cloth01 和 sample_j_colth01。如果你是个细心的人，你或许想要备份它们直到之后的基本步骤都正确完成。因为我用不着 specular map （只是件衣服用不着），我就把它删除了。（译注：作者大概漏掉了一步——将你的这两个材质里面的diffuse 改成你的贴图：j_cloth01）。

这样贴图和材质就设置好了。我发现如果你还留着那个纹章材质的话有时候载入会出现错误，所以把它删除吧。我们的材质可以以后用来做新东西的参照。

点击 Mesh 栏，点击导入，导入你最近合并过的 SMD 文件。记住把 交换Y/Z 轴（Y/Z axis swap）的勾给去掉， 现在把我们的贴图指定给导入的模型。输入材质的名字，我的是 j_cloth01（不是带sample 那个！）。

保存BRF。如果这不是mod 中的 BRF，记得把它加入mod。我们还需要再导入一次。从 heraldic_armors.brf文件夹中导入 tableau_mesh_heraldic_armor_a，参照格式给他重命名（我的是 tableau_mesh_j_cloth01），把它的材质换成你对象的材质（我的是j_cloth01）。如果你备份过的话，备份之后移除那个纹章甲。现在储存你的 BRF 文件。

（foxyman：这里介绍一个利用OpenBRF 简化上面步骤的方法。当你修改完obj 文件之后，不需要合并成smd，直接使用OpenBRF 按上述设置导入，例如你导入的文件在mod.brf 中的j_cloth01，接下来只需要打开item_meshes1.brf，点选shirt，按ctrl+c,再打开mod.brf，右键点j_cloth01，选择paste rigging，就顺利的将这个没有绑定的静态模型按原绑定转换了。关于OpenBRF ：http://bbs.mountblade.com.cn/viewthread.php?tid=82092&extra=page%3D2）

美工方面已经完成了，现在该写代码了。

12.4 网格模型（Meshes）和可变材质（Tableau_materials）

打开 module_meshes.py，搜索 tableau_mesh_heraldic_armor_a，可以找到类似东西的位置。就像大多数情况一样，我们会复制并修改它直到符合我们的需要。我的例子看上去应该像这样：

```
###JIK Heraldic (banner) armor tableau meshes
("tableau_mesh_j_cloth01", 0, "tableau_mesh_j_cloth01", 0, 0, 0, 0, 0, 0, 1, 1, 1),
```

我再重申一边，在你修改的地方加上注释。而且我不会在这教程中去修改其他高级信息，想改的话自己去学习吧。我们只需要这些就够了，现在来修改 module_tableau_materials.py，搜索heraldic_armor_d，复制下面这段代码，我们可以在这儿修改一些东西：

```
("j_cloth01", 0, "sample_j_cloth01", 512, 512, 0, 0, 0, 0,
[
(store_script_param, ":banner_mesh", 1),
(set_fixed_point_multiplier, 100),
(store_sub, ":background_slot", ":banner_mesh", arms_meshes_begin), #banner_meshes_begin),
(troop_get_slot, ":background_color", "trp_banner_background_color_array",
":background_slot"),
(cur_tableau_set_background_color, ":background_color"),
(init_position, pos1),
# (cur_tableau_add_mesh_with_vertex_color, "mesh_heraldic_armor_bg", pos1, 200, 100,
":background_color"),
(init_position, pos1),
(position_set_x, pos1, -65),
(position_set_y, pos1, 105),
(cur_tableau_add_mesh, ":banner_mesh", pos1, 0, 0),
# (cur_tableau_add_mesh, "mesh_banner_a01", pos1, 116, 0),
(init_position, pos1),
(position_set_z, pos1, 100),
(cur_tableau_add_mesh, "mesh_tableau_mesh_j_cloth01", pos1, 0, 0),
(cur_tableau_set_camera_parameters, 0, 200,200, 0, 100000),
]
),
```

按照顺序来改，首先我们需要一个唯一的ID，接下来是我们设定在 BRF 中的 material 的 sample。我不知道为什么必须得这样做，不过在弄懂“为什么不”之前还是继续如此吧。我的贴图大小本身就是 512x512，就先不改了。接下来我注释掉了定义mesh_heraldic_armor_bg 的那行。这行会给旗帜加上一些效果。之后或许还会用到它，不过也先别管它了。如果需要的话，可以把图案半你也可以在贴图留给旗帜的区域做成半透明的alpha 来取得相同的效果。向下2 行，我将pos1 设为 x-65 y105。这样旗帜就会覆盖在贴图上左边的相应位置。不清楚为什么0,0 不是某一个角上的坐标，这个坐标是我试验得出的。接下来将 cur_tableau_add_mesh 那行改为我自己的tableau 项（mesh_tableau_mesh_j_cloth01）。

如果你只要背景颜色而不要纹章图案的话，将这句注释掉：

(cur_tableau_add_mesh, ":banner_mesh", pos1, 0, 0),.

这些都完成后，就只需要添加物品并给某个人了。

12.5 物品的定义

这类物品都要用到触发器，所以为了了解这些触发器， 我们先研究一下一件已经完成的物品，搜索heraldic_mail_with，你会找到这么一段：

```
["heraldic_mail_with_surcoat", "Heraldic Mail with Surcoat", [("heraldic_armor_a",0)],
itp_merchandise | itp_type_body_armor |itp_covers_legs ,0, 3454 ,  weight(22) | abundance(100)
| head_armor(0) | body_armor(49) | leg_armor(17) | difficulty(7), imodbits_armor,
[(ti_on_init_item, [
(store_trigger_param_1, ":agent_no"),
(store_trigger_param_2, ":troop_no"),
(call_script,"script_shield_item_set_banner", "tableau_heraldic_armor_a",
":agent_no", :troop_no") ])]],
```

重要的部分就在 “,imodbits_armor” 之后，这就是触发器。不知道为什么有两个触发器参数，第二个似乎根本没用到，就别在意它了。我们只需要这部分并把它用在衬衫上。

```
["j_cloth01", "Heavy Shirt", [("j_cloth01",0)], itp_merchandise| itp_type_body_armor |itp_civilian
|itp_covers_legs ,0, 47 , weight(2) | abundance(100) | head_armor(0) | body_armor(10) |
 leg_armor(2) | difficulty(0), imodbits_cloth,
[(ti_on_init_item, [
(store_trigger_param_1, ":agent_no"),
(store_trigger_param_2, ":troop_no"),
(call_script, "script_shield_item_set_banner", "tableau_j_cloth01", ":agent_no", ":troop_no")])]],
```

唯一需要改动的是tableau 的名字，让它和你的tableauID 相对应。如果你没有在编辑模型之间的时候运行过 build_module.bat，当你第一次运行它的时候会发生错误，只要再运行它一次，错误就没了。现在把它加给你的士兵们试试吧！

12.6 低模是你的朋友

你会注意到有的 BRF 模型 有一些LOD 模型（低模），对于这个衬衫就是 shirt.lod2。早期的模型一般只有一个，不过大多数新的（尤其是有纹章的）衣甲有 lod1 至 lod3。LOD 表明了细节的等级，你观察一下会发现从原模到 lod3 下模型的细节（多边形数）在逐渐减少。不同的等级下模型的画面会相差不少，即使看上去一样，有的小地方是很难看出来的，而玩家的电脑依旧会花费效能来计算这些。

做低模既难又不难。如果你想给你的模型只配一张低模贴图的话（比如那衬衫就是基于LOD2），你得让 LOD1，LOD2 和 LOD3很相似，没什么细节。这样你就可以用LOD2或3来做。你得再UV 展开一系。如果上面这些你都会的话应该是没问题的。

为了让游戏能识别低模，你的模型必须遵循命名规则，比如我的就是 j_cloth01, j_cloth01.lod1, j_cloth01.lod2, j_cloth01.lod3。

附录(i)

附录将包含一些不是特别想在正文中注明但还是有些用的东西。
**$variable**--$变量--有着$符号的是全局变量，它们可以指编码的任何一部份，就像所有变量一样，默认值是0。
:variables--:变量--有着:符号的变量是局部变量，指只存在在它所在的编码段中。不过这些数值可以通过参数传给其他脚本。
slots--槽--所有的元组（tuple）物件都复合着槽（大约240）（foxyman注：就我所知为256个）。他们是一串默认为0 的数字。NATIVE 的可以在 module_constants.py 中看到。如果用不着槽数组的话可以去 module_constants.py 修改。把有其他用途的槽给乱改了是不好的。

Registry（foxyman注：此词应为误作，实为register） 的使用：
Reg(0)

-当一直新队伍刷出来的时候，它会先被存到 reg(0) 。（foxyman注：reg(0)或写作reg0，这样的register（寄存器）一共有64个。只要把它们看成一般的全局变量就可以了。多作为临时用途，比如在一个脚本执行完了之后将需要返回的结果存储在寄存器中。）