

从改派到 Modding

从零开始的钢铁雄心四 MOD 教程

前言

你好，我是 159753，一个普通的 mod 制作者，我个人是从“改派”，也就是修改已有的游戏/mod 文件开始走上 mod 制作之路的，所以本教程也从一个改派的视角出发，带你一步一步成为一个——会“找”的人。

什么叫“会‘找’的人”？没人会从背诵钢铁雄心 4 的所有指令开始写 MOD，相对来说更重要的是当你需要一个东西的时候，知道哪里能找到它。

好了，闲话休提，我们即刻开始。

哦对了，为了尽量照顾到更多人同时不至于过于啰嗦，本文档使用了超链接，如果你看到[这样的文字](#)，按住 Ctrl 键的同时并点击它就可以跳转到对应解释处。

第一节 文本修改

一天，你打开钢四，选择德国进入游戏……



突然之间, 你觉得“General Staff”这个名字非常不好, 你想要把它改成“Großer Generalstab”



怎么改呢?

你首先意识到一件事——如果游戏要显示什么东西, 这个东西必须存在于某处, 考虑到钢四是个单机游戏, 它显示出的“General Staff”这段话必然以某种形式存在你硬盘的某处。

幸运的是, 这是教程, 所以答案是“General Staff”以未加密字符串的形式存放在[游戏目录](#)里的一个 yml 格式文件内。

这太好了, 我们要做的仅仅是去编辑那个字符串而已……但它**具体**在哪呢?

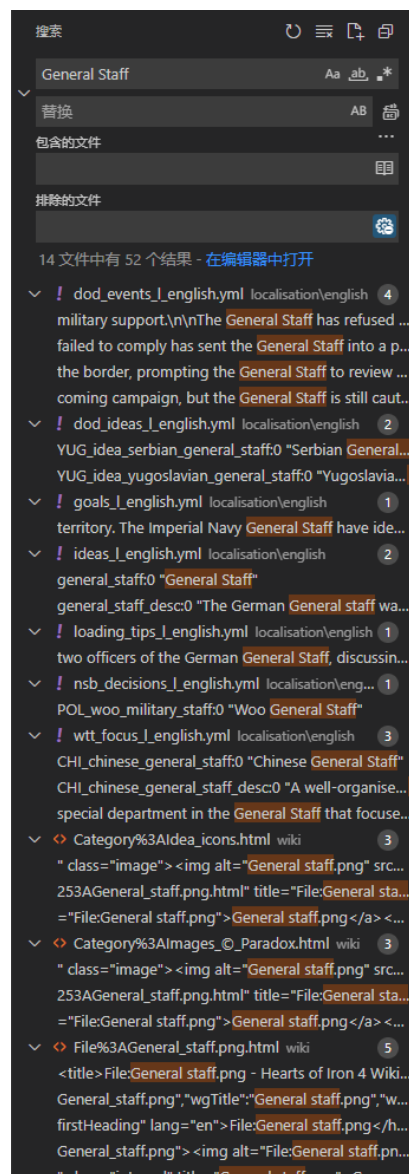
你可以选择打开游戏目录里的所有 yml 然后逐行寻找, 或者挨个打开每一个 yml 搜索对应字符串……这太慢了, 我们需要**工具**。

需要什么工具? 一个能够批量寻找某文件夹和其子文件夹内所有文件内容的软件就可以, 网上有茫茫多的软件都可以干这件事, 在本教程中我将使用

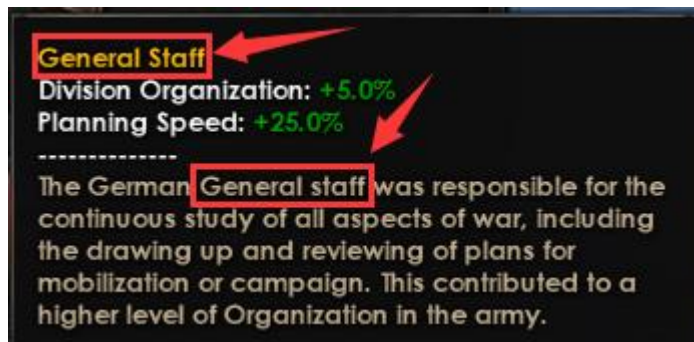
VScode 来完成这件事。(除此之外, 还有 SublimeText、Everedit、Atom、UltraEdit、Notepad++ 等等等等……总之, 如果你非要用记事本, 那么后果自负……)

注意, 以下所有操作说明都是针对 VScode 来讲的, 如果你在使用其他软件, 请自行变通。(比如 VScode 的搜索框和 Notepad++ 不在同一个位置, 但我可能把市面上每一个可能的软件的操作方法都讲一遍)

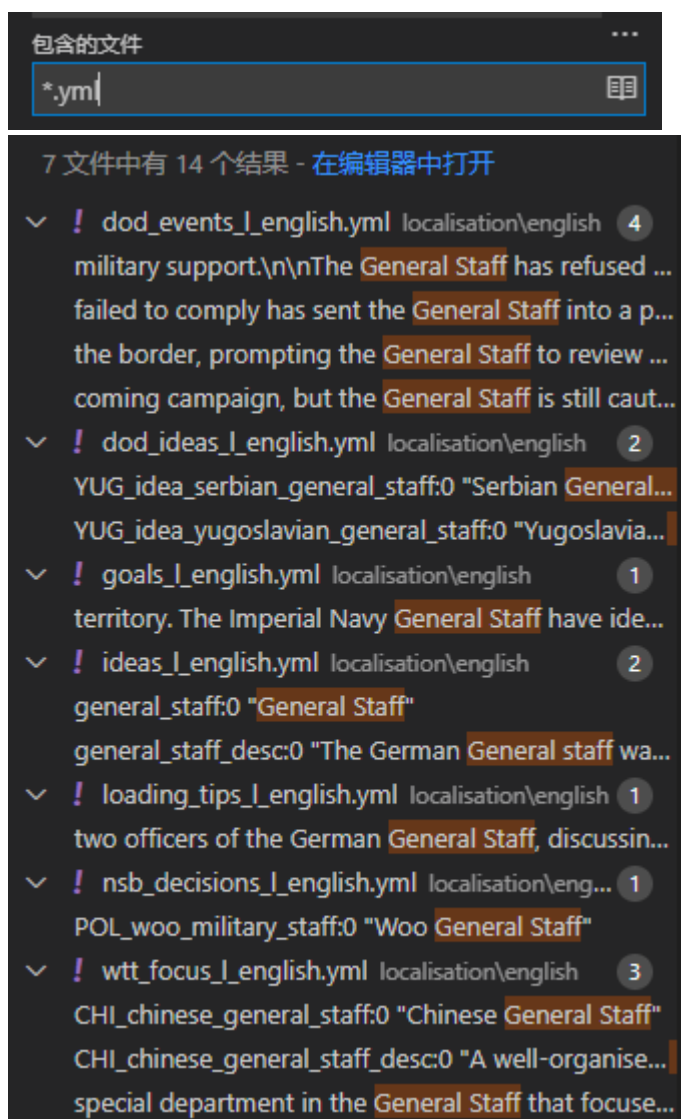
于是在一番操作之后 (指用左上角文件菜单的打开文件夹功能打开游戏目录), 你发现“General Staff”就在……



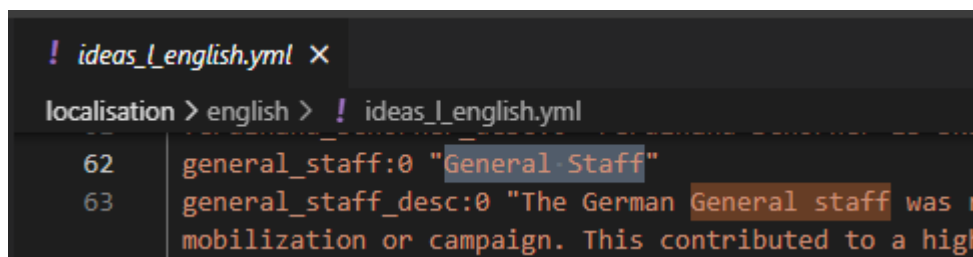
怎么会有这么多？当然了，你回想起就算是可见内容里也至少有两个“General Staff”，那么那个黄不拉几的“General Staff”到底在哪？



因为你在看教程，所以你知道钢四的文本正常情况下只存放于 yml 格式文件内，所以你选择排除所有非 yml 格式文件——（或者说只搜索.yml 文件）



还剩下 14 个结果，你又知道“General Staff”前后都没有更多文字，所以排除那些前后存在其他文字的字符串，你发现只剩下一个结果了——



```
! ideas_l_english.yml X
localisation > english > ! ideas_l_english.yml
62 general_staff:0 "General Staff"
63 general_staff_desc:0 "The German General staff was r
mobilization or campaign. This contributed to a high
```

啊哈！“General Staff”就保存在游戏目录内 localisation 文件夹的 ideas_l_english.yml 文件夹内！钢四游戏修改不过如此！选中这一行……

停！

注意你现在打开的是**游戏原文件**，这次我们只改了个字符串，如果下次你打算改一些其他的東西，改完游戏出了 BUG 或者干脆不启动了怎么办？

所以说，无论是当改派还是制作 mod，第一条铁律：

永远备份源文件

其实这不怎么铁，但就好像“闻试剂时不要直接鼻孔对着瓶口”这种规矩一样，不遵守不一定会出事，但要出事了……（就会害得你重装游戏，对于后一个规则，可能是人生速通）

所以我们先来讲一讲[如何创建一个 mod](#)。（和正文关系不大，请按住 Ctrl 点击左边链接浏览）

好，假设你已经按照上述教程创建好了一个 mod，接下来我简单介绍一下钢四 mod 对游戏修改的规则——同路径同文件名替换，或者用人话说，加载一个 mod 就相当于将 mod 文件覆盖到游戏目录内。

那么我们上面已经知道“ ‘General Staff’ 就保存在游戏目录内 localisation 文件夹的 ideas_l_english.yml 文件夹内”，现在只需要在 mod 目录内创建一个名为 localisation 的文件夹，再将 ideas_l_english.yml 复制到这个位置打开，将其中的 general_staff:0 "General Staff"改为 general_staff:0 "Großer Generalstab"

你问为啥不直接创建个新的 ideas_l_english.yml？因为原版的 ideas_l_english.yml 里还存着许多其他字符串，你用只含有一行 general_staff:0 "Großer Generalstab"的文件去替换，剩下其他内容就全丢失了。

做完以上一切，打开启动器，启用你刚刚创建的 mod，进入游戏，大功告成！



小结

无论是原版游戏还是 Mod 显示出的绝大部分文本都保存在对应目录 localisation 文件夹里的 yml 文件内，也就是本节中提到的方法不仅能修改国家精神（National Spirit/ideas），也能修改各类事件名、事件描述和国名等，或者说得不严谨一点——可见即可改。

第二节 Mod 文本修改

有那么一个钢四中文玩家喜闻乐见的 Mod, 它毫无疑问是钢四中文玩家使用次数最多的 Mod, 它是——52pc 汉化 Mod (下文简称汉化 Mod), 今天我们就来修改它的内容。

今天你开启汉化 Mod 进入游戏, 忽然觉得“总参谋部”应该改为“德国总参谋部”



怎么改呢?

用第一节学到的知识, 去游戏目录内执行搜索, 发现连一个中文字符都搜不到, 自然, 钢四没有官方中文也不支持显示中文。(所以汉化 mod 自带一个中文字库, 但这方面的知识我们暂时按下不表)

当然, “总参谋部”不在游戏目录里就肯定在别的什么地方——汉化 mod 的文件目录里, 在哪?

因为你在看教程, 所以你知道钢四读取任何 Mod 都要先读取引导文件, 而所有引导文件都在 mod 存放目录里, 于是你在 mod 存放目录里使用全局搜索功能顺利找到了汉化 mod 的引导文件——“ugc_698748356.mod” (其 name 段内容是“52 Chinese Localisation”, 也就是 Mod 名称), 可 mod 文件又在哪呢? 你

想起上一节我们提到过引导文件内会指出 mod 文件的位置，打开 ugc_698748356.mod，找到 path 行，有：

“F:/SteamLibrary/steamapps/workshop/content/394360/698748356”

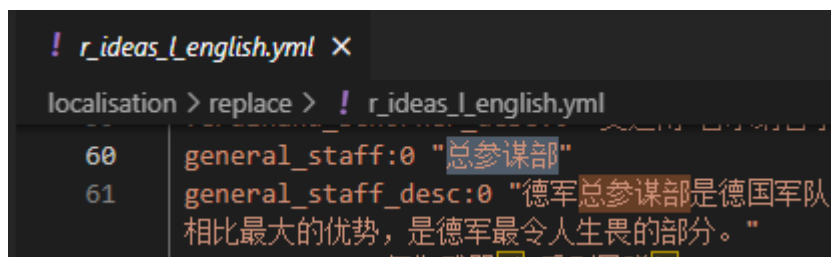
（这也就是上节所说“path 不指向 mod 存放目录内”的情况，注意你得到的具体路径仍取决于其他因素，比如我的 steam 库位置是 F:/SteamLibrary 所以我得到的路径如上，你的库路径不一样得到的路径就不一样）

聪明的你立刻意识到“698748356”这个数字恰好代表着这个 Mod 的创意工坊编号。（<https://steamcommunity.com/sharedfiles/filedetails/?id=698748356> 则可直接访问汉化 Mod 的发布页），而 394360 是钢铁雄心 4 这款游戏的 AppID。

于是访问 F:/SteamLibrary/steamapps/workshop/content/394360/698748356，我们看到了熟悉的 Localisation 文件夹，搜索之：



这太简单了，你很快找到：



但还记得第一节我们走到这一步的时候发生了啥吗？

停，**备份原文件**。

对于 Mod 来说，你可以简单的将这个 mod 的所有内容都一股脑复制到一个你创建的新 Mod 目录中，接下来继续修改。

将 general_staff:0 "总参谋部"改为 general_staff:0 "德国总参谋部"，保存，关闭汉化 Mod，开启你的新 Mod，大功告成。



小结

和上一节一样，只不过我们这一次修改了非原游戏文件，相信你已经不满足于修改文本（或者本来就是冲着改游戏效果来的），那么下一节我们学习修改效果。

第三节 效果修改

这一次，你又觉得总参谋部提供的加成太小了，怎么才 5%的陆军组织度？我要给它翻个倍，10%！

但是某个国家精神提供的加成存在哪里呢？不在 yml 里，你找不到相应的“陆军组织度: +5%”或“Division Organization: +5%”的条目。

思考中，你意识到一件事，为什么在 yml 里找到的条目都是以“xxxx:0 “yyyy” ”的形式组织起来的呢？我们一直在修改的都是 yyyy，那 xxxx 是做什么用的？

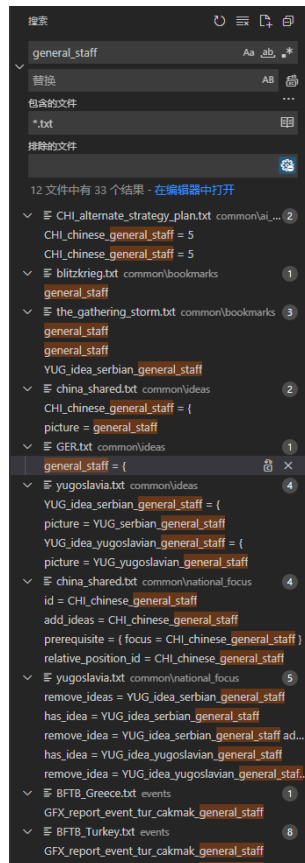
xxxx 就是所谓的“键值”（key value），或者说人话，“内部名”，为什么需要内部名？内部名就好像身份证，你可以找到无数个重名的人，但身份证号可以重复吗？如果重复了我们在身份证系统里怎么区分两个人？

所以键值/内部名就是游戏元素（国家、国家精神、事件等等）的身份证号，它是唯一的，而 yml 所做的事情就是给这些内部名一个“外号”，将一个可以重复的显示名（姓名）分配给一个唯一的内部名（身份证号）。

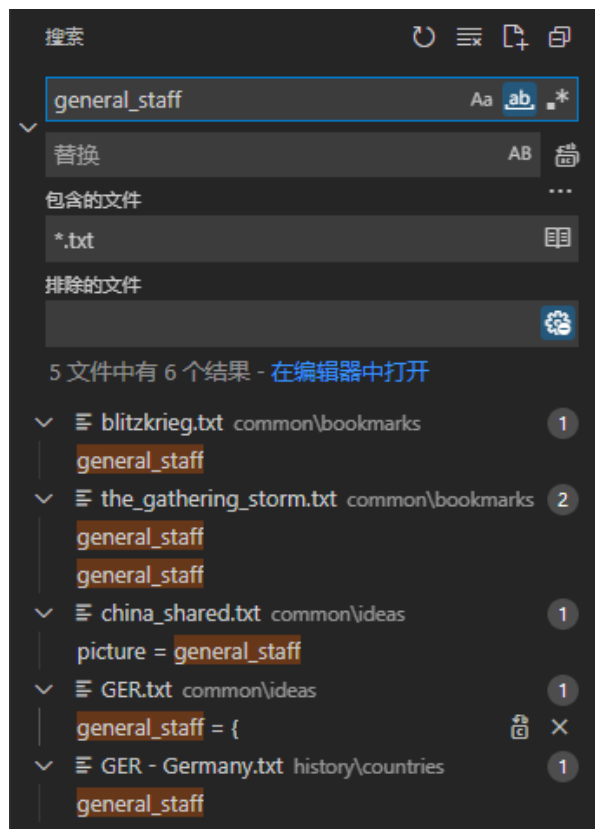
这里最生动的例子就是原版钢四的梅福票，为什么梅福票每隔一段时间提供的加成都会变化？实际上游戏是把一个梅福票国家精神换成了另一个加成不同但同样也叫“梅福票”的新国家精神。（只不过这个操作在游戏层面上是瞬间完成的，所以你看不出）

那么，general_staff:0 “总参谋部”这句话的意思其实就是：“[让 general_staff 这个国家精神的名字显示为总参谋部](#)”，既然内部名唯一，我们去找 general_staff 不就好了？

用同样的方法，同时因为这是教程，你排除了非 txt 文件（钢四的国家精神和绝大部分玩意都用 txt 存，就是这么朴实无华），搜索，得到——



怎么这么多？你赶紧切换为全词匹配（前后不得有其他字符）。



怎么回事？说好的唯一呢？

正准备骂教程作者净瞎胡扯，你突然发现尽管 `general_staff` 全词匹配也会得到六个结果，但其中只有一个在其后有“`= {`”，这是什么意思呢？该不会“`= {`”的意思类似中文里的“：”？

对了，“`= {`”在这里就类似“定义”，其他的匹配结果其实只是在使用 `general_staff` 这个“身份证号”，而这个身份证号要携带什么内容——你打开了 `common\ideas\GER.txt`，找到对应位置：

```
general_staff = {  
    allowed = {  
        original_tag = GER  
    }  
  
    allowed_civil_war = {  
        always = yes  
    }  
  
    removal_cost = -1  
  
    modifier = {  
        army_org_factor = 0.05  
        planning_speed = 0.25  
    }  
}
```

哇，`general_staff` 这个玩意的一对大括号里居然包含了足足四个项目，而其中 `modifier` 项里还包着四个参数。

不要怕，我知道你英语过了四级，什么？没过？那你肯定有网，要不然也看不到这篇教程，无论如何，你在人工翻译/机器翻译之后理解了一切。

`allowed`，作为形容词时代表“合法的”，那么 `allowed` 的大括号里包着的是否代表“何时合法”？我们看 `original_tag = GER` 这一项……嘿，还记得吗，**你有网哎！**

所以与其在这里机翻瞎猜，为什么不看看现成的资料？

https://hoi4.paradoxwikis.com/Idea_modding 这是钢铁雄心 4 英文维基的对应

页面，打开，翻到 Arguments 一栏：

Arguments [编辑 | 编辑源代码]

- allowed** determines whether or not a country gets this idea or not. It is only checked at the game's start and never again. It's especially important for ministers and designers since not putting an allowed will mean that every country in the game will have access to that designer or minister. For most ministers and designers, `original_tag = XXX` is generally enough, since you want a country (and possible rebellions) to have access to that minister or designer. `add_ideas` ignores this check, so it does nothing in splits or hidden ideas.
- allowed_civil_war** is a variant of allowed that determines if an idea will be transferred to a rebelling country during a civil war. Most of the time it's used for ideology checks to ensure only one side has that idea in a civil war.
- available** decides if a country can select it. If a country does not meet the requirements, then it will not be able to select it, and it will be automatically removed if the country already has it.
- visible** decides when an idea is visible and can be picked.
- cancel** will automatically remove the idea if the conditions inside are met.
- picture** decides the picture used by the idea. The picture is defined in `/Hearts of Iron IV/interface/*.gfx`. An example definition looks like this:

```
spriteType = {
    name = "gfx_idea_picture_name"
    textureFile = "gfx/interface/ideas/<file name>.dds"
}
```

Note that file name and the sprite's name do not need to match. This sprite is used in the idea as `picture = <picture name>`. **Note that GFX idea is automatically inserted by the game and shouldn't be used in the picture argument**

It is also possible to make the picture used to depend on the graphical_culture_2d of the country, defined in `/Hearts of Iron IV/common/countries/*.txt`. Such a definition looks like:

```
spriteType = {
    name = "gfx_idea_picture_name.<graphical culture's name>"
    textureFile = "gfx/interface/ideas/<file name>.dds"
}
```

If a picture is not specified, the game will set `gfx_idea_idea_name` to be used by default.

- ledger** is used to assign ideas to the intelligence ledger. It is necessary to be used in ideas that are defined in categories that have the 'invalid' ledger defined, which are 'theorist' and 'high_command' in the base game. If the idea slot has a ledger defined, the idea's individual ledger will take priority. Options include army, navy, air, civilian, and hidden.
- modifier** assigns the modifiers to the country that has them. Modifiers are listed one after another in a modifier's bracket. The list of modifiers can be seen on the [modifiers](#) page. Here's an example of an idea that would increase a country's Recrutable Population as well as lowering its factory output:

机翻一下：

Arguments [编辑 | 编辑源代码]

- allowed**决定了一个国家是否有这个想法。它只在游戏开始时被检查，永远不会再被检查。这对部长和设计师来说尤其重要，因为没有允许将意味着游戏中的每个国家都可以访问该设计师或部长。对于大多数部长和设计师来说，`original_tag = XXX` 通常就足够了，因为您希望一个国家（以及可能的叛乱）可以访问该部长或设计师。`add_ideas` 忽略这个检定，所以它不会对精神或隐藏的想法产生任何影响。
- allowed_civil_war**是 `allowed` 的一个变体，用于确定一个想法是否会在内战期间转移到一个叛乱国家。大多数时候，它被用于意识形态检查，以确保在内战中只有一方有这个想法。
- available**决定一个国家是否可以选择它。如果一个国家不符合要求，那么它就无法选择它，如果这个国家已经有它，它会被自动删除。
- 可见**决定一个想法何时可见并且可以被挑选。
- 如果满足内部条件，**取消将自动删除该想法**。
- 图片**决定了idea使用的图片。图片在`/Hearts of Iron IV/interface/*.gfx` 中定义。示例定义如下所示：

好吧，很无聊，而且看起来根本和加成没关系，继续机翻……

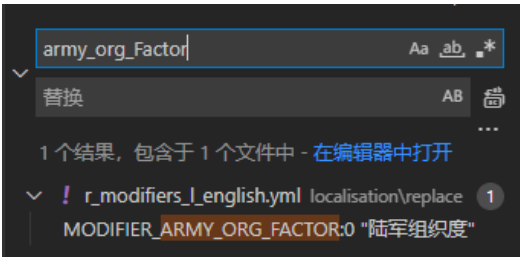
...

- 修饰符**将修饰符分配给拥有它们的国家。修饰符在修饰符的括号中一个接一个地列出。修改器列表可以在[修改器](#)页面上看到。这是一个可以增加一个国家的可招募人口并降低其工厂产量的想法示例：

修饰符？你切回英文版,发现这对应 modifier 一词,而我们在之前的 general_staff 里发现了这么一块：

```
modifier = {
    army_org_Factor = 0.05
    planning_speed = 0.25
}
```

你已经猜到了这里的 0.05 转换为百分数是 5%，而 0.25 转换后对应 25%，恰好就是总参谋部两个加成的数值，你又在汉化 mod 的文件里去搜索 `army_org_Factor`：



果然，`army_org_Factor` 就是“陆军组织度”这个加成的“身份证号”（键值），而 `planning_speed` 则是计划速度的键值，事情到这里已经了然，你将 `army_org_Factor = 0.05` 改为……

不要直接修改原文件！

于是你将 `GER.txt` 复制了一份到你自己的 `mod` 的 `common\ideas` 目录下，将 `army_org_Factor = 0.05` 改为 `army_org_Factor = 0.1`，保存，启用 `mod`，进入游戏。



完成！

小结

这次我们进行了一次最简单的数值修改，但其中用到的方法是极重要的，我将其称之为“**顺藤摸瓜**”，既然 `yml` 文件是完成身份证号(键值)→姓名(显示名/本地化名)转化的关键，那么只要知道两者中的其中一个，我们就能利用 `yml` 文件作为中介找到另一半。

下一节中，我们将进一步使用这种方法。

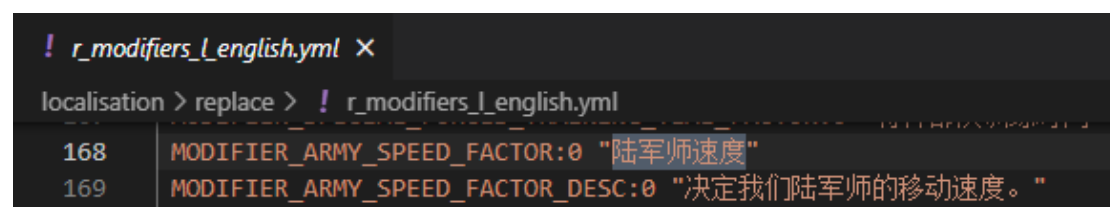
第四节 进阶效果修改

在爽完了组织度无限或计划爆炸的雅利安超人之后，你陷入了空虚……总参谋部的加成不能满足你球长的需要，你想要更多，比如总参谋部能让德军士兵双腿赛过喷气机怎么样？

于是你打开游戏……有什么东西能给部队加速呢？机动作战的第一个学说可以！



“陆军师速度”就是我们要找的加成，注意现在这个加成名称是汉化 mod 提供的，所以我们要去汉化 mod 的文件夹内寻找它：



不知道你有没有好奇，上一节我们搜索 `army_org_Factor` 的本地化名时，找到的分明是 `MODIFIER_ARMY_ORG_FACTOR` 但我却直接说 `army_org_Factor` 对

应的就是“陆军组织度”？

这就牵扯到[更进一步的本地化——键值转换](#)了，目前你只需要知道形如 MODIFIER_AAAAA 的加成键值在实际使用时要转化为 aaaaa（去掉 MODIFIER_，变为小写）即可。

所以我们将 army_speed_factor = 1 加入到 general_staff 的 modifier 一栏内：

```
general_staff = {  
    allowed = {  
        original_tag = GER  
    }  
  
    allowed_civil_war = {  
        always = yes  
    }  
  
    removal_cost = -1  
  
    modifier = {  
        army_org_factor = 0.05  
        planning_speed = 0.25  
        army_speed_factor = 1  
    }  
}
```

启用 mod，打开游戏：



你是否也好奇为什么我们明明把 `army_speed_factor = 1` 加到了 `modifier` 的最后一项，显示的时候陆军师速度却跑到了第一位？

这跟它们在 `yaml` 里的排序有关，但这不会影响实际执行效果，无需在意。

小结

本节中我们小小地实践了一下顺藤摸瓜法，我想特别强调的是，顺藤摸瓜法，或者说利用本地化名寻找键值（反之亦然）的方法是**通用性极高**的，不仅仅在修改一个小小的国家精神上可以适用。

下节中我们将证明这一点。

第五节 顺藤摸一切

这一节我们会略微提高一点难度，改一改“功能性”mod，以创意工坊的“少前+碧蓝将领培养系统+新单位与机制”

(<https://steamcommunity.com/sharedfiles/filedetails/changelog/2453634171>)

mod 为例……直接在教程里改他人 mod 合适吗？没有关系，我就是该 mod 的作者，所以有授权（笑）。

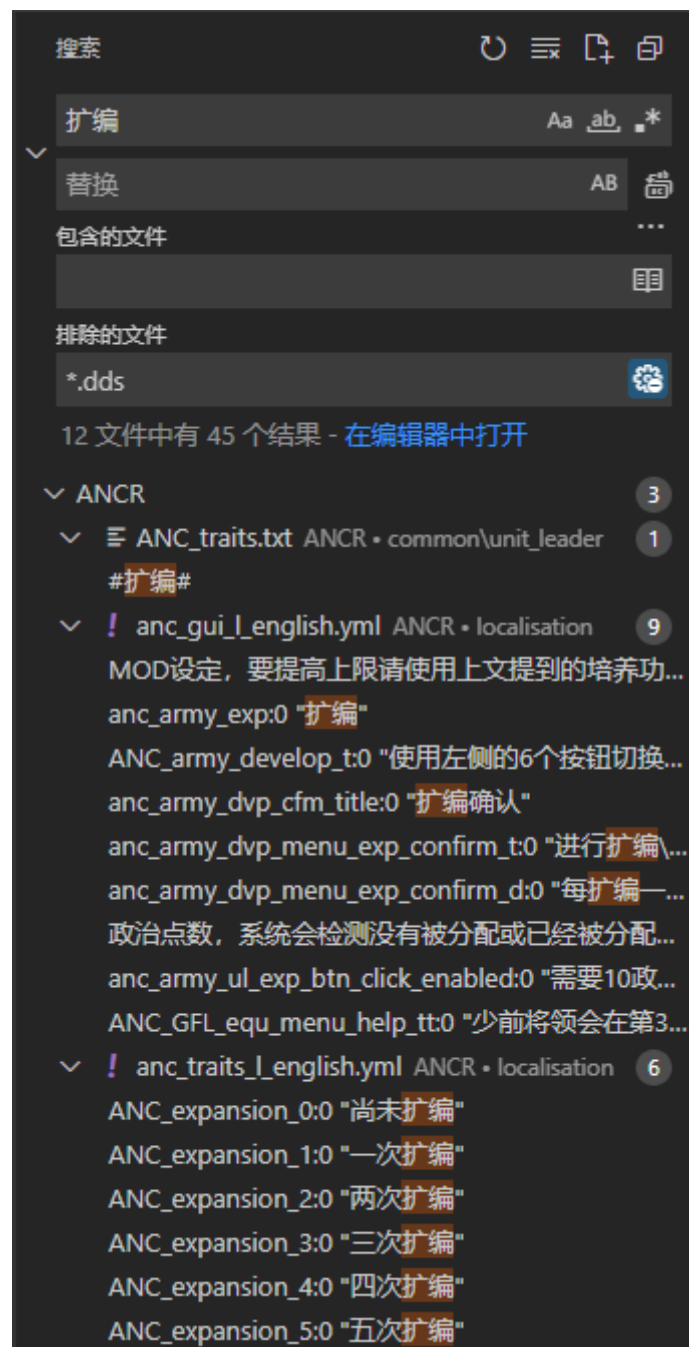


该 mod 的将领最初只能带 3 个师，需要在领袖界面里“扩编”才行——

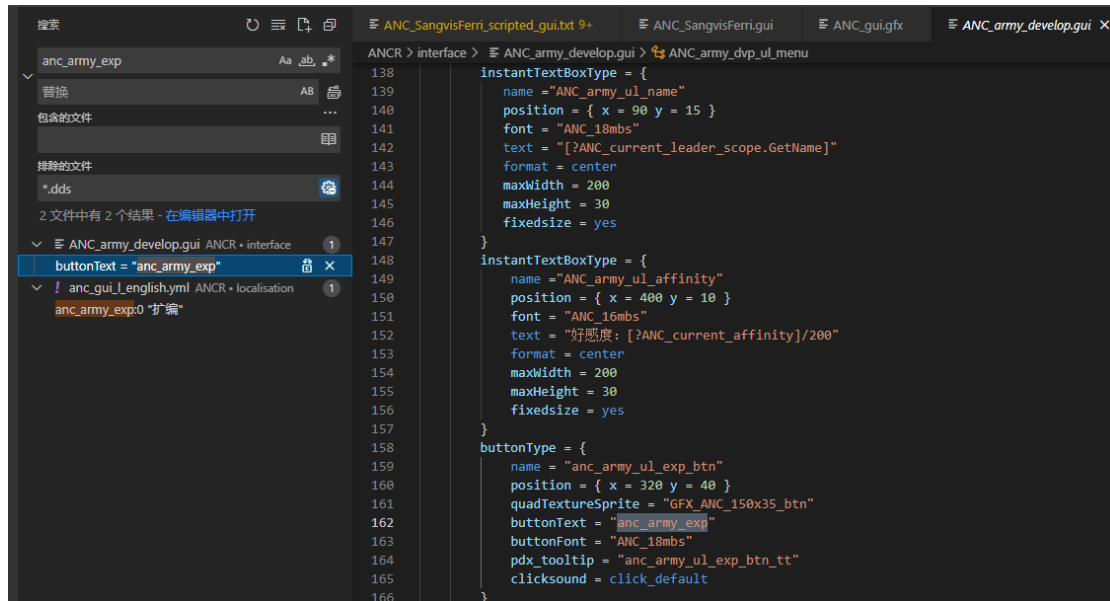


可是这要消耗 10 政治点，而且扩编满需要点击足足 5 次！太麻烦了！现在我们要让它点一次就扩编满，并且去除所需 10 政治点这一限制条件。

乍一看起来这简直太困难了，我们要修改一个通过自定义 gui 实现的东西——但是不用怕，想想我们在教程最开始知道的第一件事是什么？“如果游戏要显示什么东西，这个东西必须存在于某处”，那么我们知道扩编这件事需要通过点击一个上面显示着“扩编”二字的按钮实现……



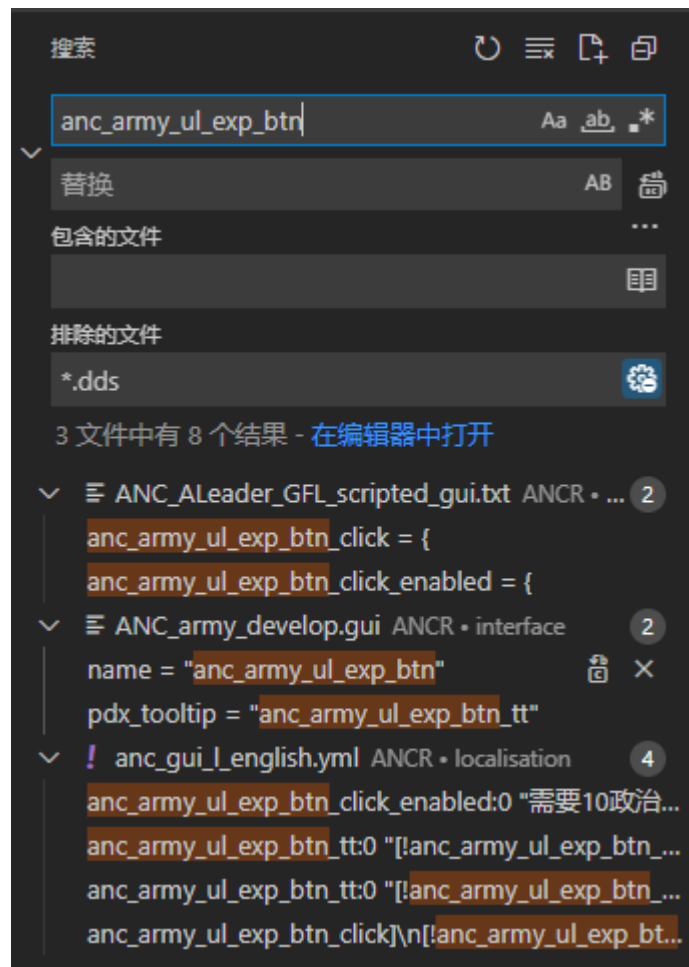
非常好，只有一个结果，过去看一看，内部名为“anc_army_exp”，再度搜索该名称：



ANC_army_develop.gui? 怎么是个拓展名为 gui 的文件？别着急，**我们有网**。

你可以选择浏览 wiki 上的[对应页面](#)，或者发挥一下“聪明才智”——我们已经知道，钢四的代码一般以 `XXX = { YYY = "ZZZ" }` 的形式组织，XXX 一般为本体，而 YYY 则为 XXX 的属性，ZZZ 是 YYY 的值，这样一来，再观察我们得到的结果，可以得出“anc_army_exp”是 buttonText 的值，而 buttonText 又是 buttonType 的一个属性……

我们要搜索 buttonType 吗？不，我们能发现该文件内出现过很多个 buttonType，那么是什么把这些区分开的呢？名字？是的，名字，或者说 buttonType 的 name 属性是区分它们的关键，那么我们来搜索一下“anc_army_ul_exp_btn”：



由于 yml 文件里只储存着本地化名（外号），因此 yml 文件里的结果可以无视，而 gui 文件是我们的来路，唯一的结果只剩下 ANC_ALeader_GFL_scripted_gui.txt 这个文件了，进入：

```

effects = {
    anc_army_ul_exp_btn_click = {
        var:ANC_current_leader_scope = {
            if = {
                limit = { has_trait = ANC_expansion_0 }
                remove_trait = { trait = ANC_expansion_0 }
                add_trait = { trait = ANC_expansion_1 }
                log = "exp1"
            }
            else_if = {
                limit = { has_trait = ANC_expansion_1 }
                remove_trait = { trait = ANC_expansion_1 }
                add_trait = { trait = ANC_expansion_2 }
                log = "exp2"
            }
            else_if = {
                limit = { has_trait = ANC_expansion_2 }
                remove_trait = { trait = ANC_expansion_2 }
                add_trait = { trait = ANC_expansion_3 }
                add_to_variable = { ANC_GFLul_available_eqpslot = 1 }
                log = "exp3"
            }
            else_if = {
                limit = { has_trait = ANC_expansion_3 }
                remove_trait = { trait = ANC_expansion_3 }
                add_trait = { trait = ANC_expansion_4 }
                add_to_variable = { ANC_GFLul_available_eqpslot = 1 }
                log = "exp4"
            }
            else_if = {
                limit = { has_trait = ANC_expansion_4 }
                remove_trait = { trait = ANC_expansion_4 }
                add_trait = { trait = ANC_expansion_5 }
                add_to_variable = { ANC_GFLul_available_eqpslot = 1 }
                log = "exp5"
            }
        }
        add_political_power = -10
        hidden_effect = {
            meta_effect = {
                text = {
                    | scoped_sound_effect = ANC_COMBINE_sound_[GFLname]
                }
                GFLname = "[Get_ANC_leader_name]"
            }
        }
    }
}

```

好家伙！直播事故！（是的，笔者写到这里之后发现这个功能没那么简单，没法三两句话就完全讲清楚，但放心，改还是能改的）

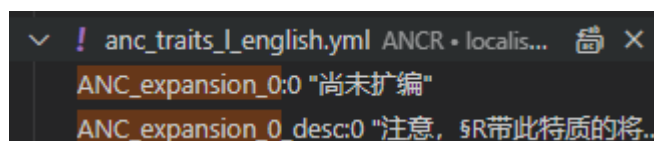
看着眼花缭乱的代码，你决定先从可以理解的部分下手——

通过浏览 [wiki \(该处有网址\)](#)，你能知道 `add_political_power` 是添加政治点的效果，而图中使用了 `add_political_power = -10`，增加负十就等于减十，那看来这块便是实现减少政治点的地方了，同时还能知道 `add_trait = { trait = ANC_expansion_3 }` 这个效果代表着添加一个名称为 `ANC_expansion_3` 特质……

再加上对 `if` 和 `else_if` 的翻译，你可以部分解读出这段代码的意思：

“如果有 `ANC_expansion_0` 特质，则移除 `ANC_expansion_0` 特质，添加 `ANC_expansion_1` 特质，不然的话，判定是否有 `ANC_expansion_1` 特质，如果有，移除 `ANC_expansion_1` 特质，添加 `ANC_expansion_1` 特质，还不满足条件，判定是否有 `ANC_expansion_2` 特质……添加-10 点政治点数……”

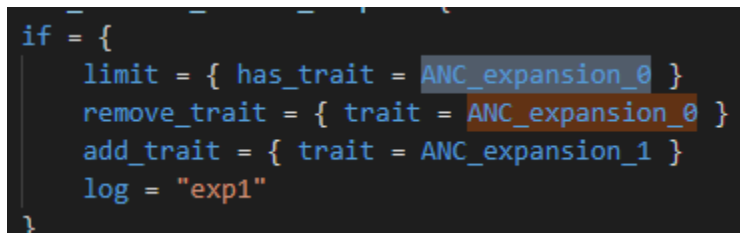
这个时候，我们能够意识到所谓的五次扩编是通过逐次移除上一级特质并添加下一级特质完成的，为了验证这个想法，我们可以搜索一下 `ANC_expansion_0`：



```
! anc_traits_l_english.yml ANCR • localis...  
ANC_expansion_0:0 "尚未扩编"  
ANC_expansion_0_desc:0 "注意，5R带此特质的将..."
```

验证完毕，那么要一次到顶，就应该让这段代码无论如何都在第一次判定时就添加最后一级也就是 `ANC_expansion_5` 特质，如何编写呢？

我们知道第一次判定位于这个位置：



```
if = {  
  limit = { has_trait = ANC_expansion_0 }  
  remove_trait = { trait = ANC_expansion_0 }  
  add_trait = { trait = ANC_expansion_1 }  
  log = "exp1"  
}
```

如果通过则移除 `ANC_expansion_0` 并添加 `ANC_expansion_1`，如果不通过则转到下一个 `else_if` 进行判定，那就很简单了，我们将通过条件 (`has_trait =`

ANC_expansion_0) 设为永远通过 (always = yes) ……哎? always = yes 是啥? 是你能在 [wiki 上找到的东西](#)。

将效果 (add_trait = { trait = ANC_expansion_1 }) 修改为添加
ANC_expansion_5 (add_trait = { trait = ANC_expansion_5 }), 大功告成!

小结

本节中, 我们进一步修改了代码, 不知道你发现没有, 在这一节里我们遇到的很多东西我都没有解释, 但修改仍然成功了。

这既是顺藤摸瓜法的优点, 却也是它的缺点, 它无法让你总揽全局, 只能“知其然”, 不能“知其所以然”, 因此下一节中本教程将不可避免的变无聊, 因为我们要开始谈一些不那么直观的东西了。

第六节 找规律游戏

有这么一串字母：AAABBBAAACCCAAADDDAAAEEEEAAFFFAAAGGG……

请问，接下来的六个字母最有可能是？

A、AADDCC

B、DDDDAA

C、AAHHHH

D、AAASDS

如果你能看出来这道题的答案是 C，那么恭喜你，你已经具备了可以说是完全的编写钢四 mod 的能力。

让我们把它变成“钢四 mod 制作风格”再试一次——

这是一些原版的国家精神文件切片：

```
sour_loser = {  
    allowed = {  
        | always = no  
    }  
  
    allowed_civil_war = {  
        | always = yes  
    }  
  
    removal_cost = -1  
  
    modifier = {  
        | drift_defence_factor = 0.5  
    }  
    rule = {  
        | can_create_factions = yes  
    }  
}
```

```
general_staff = {  
    allowed = {  
        | original_tag = GER  
    }  
  
    allowed_civil_war = {  
        | always = yes  
    }  
  
    removal_cost = -1  
  
    modifier = {  
        | army_org_Factor = 0.05  
        | planning_speed = 0.25  
    }  
}
```

```
triumphant_will = {
    allowed = {
        always = no
    }
    removal_cost = -1

    allowed_civil_war = {
        has_government = fascism
    }

    modifier = {
        political_power_gain = 1
        drift_defence_factor = 0.5
    }
    rule = {
        can_create_factions = yes
    }
}
```

```
GER_rocketry_idea = {
    removal_cost = -1
    allowed = {
        always = no # Unlocked via focus
    }

    allowed_civil_war = {
        always = yes
    }

    research_bonus = {
        rocketry = 0.15
    }

    picture = generic_research_bonus

    modifier = {
        production_speed_rocket_site_factor = 0.15
    }
}
```

```
GER_ostwall_idea = {
    allowed = {
        original_tag = GER
        always = no
    }

    allowed_civil_war = {
        always = yes
    }

    removal_cost = -1

    picture = generic_wall_line

    modifier = {
        production_speed_bunker_factor = 0.2
    }
}
```

```
GER_grosraumwirtschaft = {
    allowed = {
        original_tag = GER
        always = no
    }

    allowed_civil_war = {
        always = yes
    }

    removal_cost = -1

    picture = generic_build_infrastructure

    modifier = {
        production_speed_infrastructure_factor = 0.1
        production_speed_rail_way_factor = 0.1
    }
}
```

现在，请问以下哪个图片里的代码最有可能符合格式？

```
OPTION_A = {
    allowed = {
        WhatTriggerIsThis = ?
    }

    allowed_civil_war = {
        always = yes
    }

    removal_cost = -1

    picture = generic_build_infrastructure

    modifier = {
        production_speed_infrastructure_factor = 0.1
        production_speed_rail_way_factor = 0.1
    }
}
```

```
OPTION_B = {
    allowed = { always = no }

    allowed_civil_war = { always = no }

    removal_cost = -1

    picture = generic_build_infrastructure

    modifier = {
        production_speed_rail_way_factor = 1
    }
}
```

```

OPTION_C = {
  allowed = {
    original_tag = GER
    always = no
  }

  allowed_civil_war = {
    always = yes
  }

  removal_cost = -1

  picture = generic_build_infrastructure

  modifier = {
    whereIsThisModifier = 0.22
    production_speed_infrastructure_factor = 0.1
    production_speed_rail_way_factor = 0.1
  }
}

```

```

OPTION_D = {
  allowed = {
    original_tag = GER
    always = no
  }

  allowed_civil_war = {
    always = yes
  }

  removal_cost = -1

  picture = generic_build_infrastructure

  modifier = {
    production_speed_infrastructure_factor = 0.1
    production_speed_rail_way_factor = 0.1
  }
}

```

答案解析:

```

OPTION_A = {
  allowed = {
    WhatTriggerIsThis = ?
  }

  allowed_civil_war = {
    always = yes
  }

  removal_cost = -1

  picture = generic_build_infrastructure

  modifier = {
    production_speed_infrastructure_factor = 0.1
    production_speed_rail_way_factor = 0.1
  }
}

OPTION_B = {
  allowed = { always = no }

  allowed_civil_war = { always = no }

  removal_cost = -1

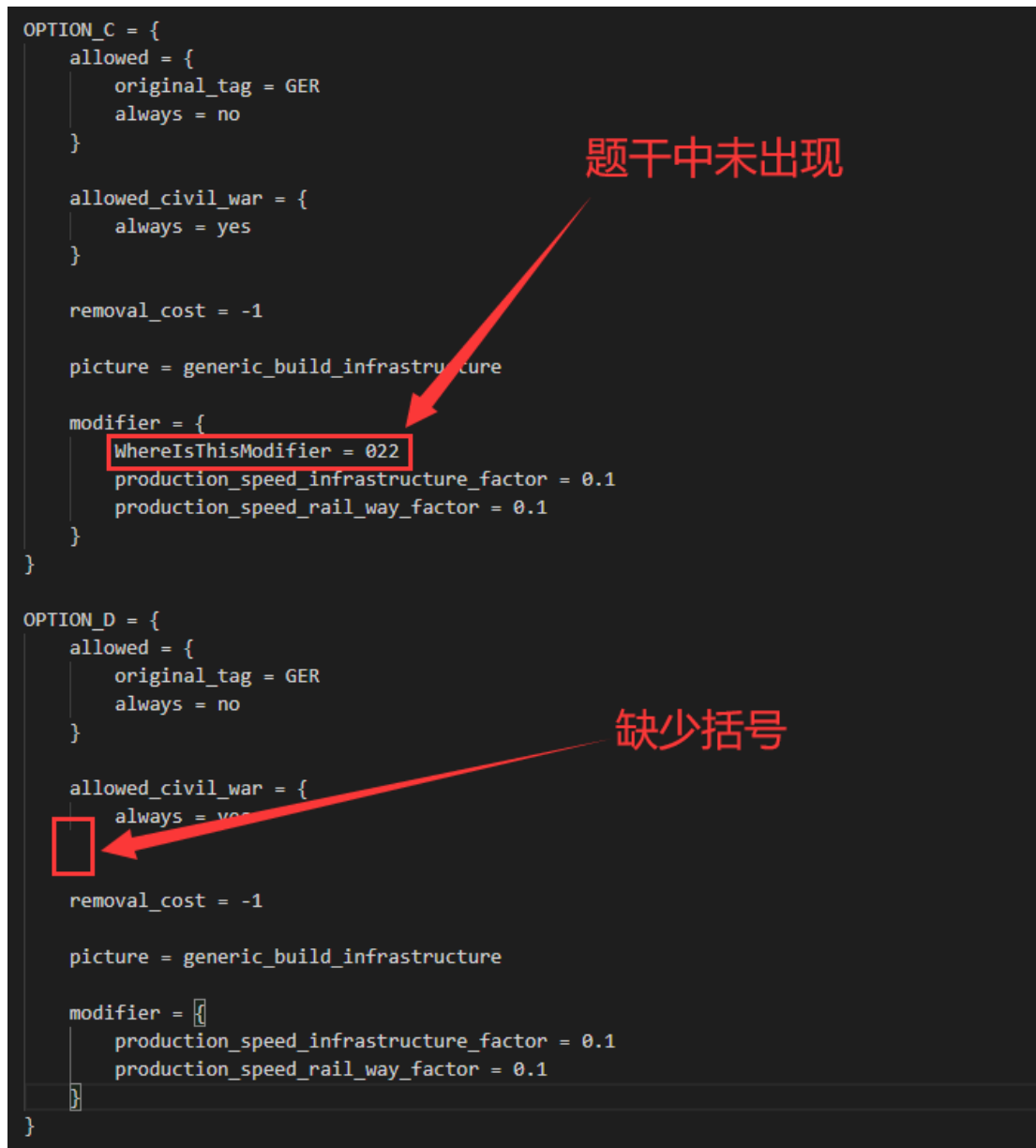
  picture = generic_build_infrastructure

  modifier = {
    production_speed_rail_way_factor = 1
  }
}

```

题干中未出现

✓



每当你试图添加一个新东西，无论那是国家、单位、国家精神还是什么其他的，你都在做这么一道找规律题，题干就是所有相关的原版内容。

试问，如果你像上题选项 A 那样在 `allowed = { }` 内增加了一个 `WhatTriggersThis = ?` 项，而找遍整个游戏都没有这么一项的身影……它还会是正确的吗？

记住，你在制作 MOD，modification，修改，而不是创造，游戏里没有的代码就是没有，不要发明。

小结

每当你完成一段代码，问自己三个问题：

0、这个写法我在原版里见过吗？

1、这个写法我在原版里见过吗？

2、这个写法我在原版里见过吗？

发明，是 P 社程序员的事情，不是我们的事情。

第七节 什么叫写法

我们在第五节里提到过，钢四（可以被我们看到的部分）代码一般以 XXX = { YYY = ZZZ } 的格式出现，但这太笼统了，而且不同的内容有着不同的组织形式……第六节中的那道题就有无数个正确答案，尽管答案之间存在着确定的规律，但作为有互联网的人，我们无需亲自总结规律，看别人总结好的就可以了。

<https://hoi4.paradoxwikis.com/Modding>

Modding	
Documentation	Effects • Triggers • Defines • Modifiers • List of modifiers • Scopes • Localisation • Variables • Arrays • On actions
Scripting	AI • AI Focuses • Autonomous states • Bookmarks • Buildings • Characters • Cosmetic tags • Countries • Divisions • Decisions • Equipment • Events • Ideas • Ideologies • National Focuses • Resources • Scripted GUI • Technology • Units
Map	Map • States • Supply areas • Strategic regions
Graphical	Interface • Graphical Assets • Entities • Posteffects • Particles • Fonts
Cosmetic	Portraits • Namelists • Music • Sound
Other	Console commands • Troubleshooting • Mod structure • Mods

尽管这不是官方 wiki，但它可以解决基本 95% 的 mod 制作问题，譬如说我们要添加一个国家精神，不想做找规律题，怎么办？

https://hoi4.paradoxwikis.com/Idea_modding

Note that positives make technology faster to research while negatives make it slower.

- **rule** assigns the game rules changed to the country by that idea.

Allowed game rules are `can_be_called_to_war`, `can_boost_other_ideologies`, `can_create_factions`, `can_decide_to_start_war`, `can_generate_female_aces`, `can_guarantee_other_ideologies`, `can_join_factions`, `can_join_factions_not_alone`, `can_only_justify_war_on_threat_country`, `can_use_kamikaze_pilots`, and `units_deployed_to_overlord` (for `scripted_gui`).

- **traits** assign a trait to an idea. They will show up when hovering over the idea and they will apply the effect to the country leader. The file is located at `IV/common/country_leader/*.txt`.
- **on_add** assigns effect to be executed when adding the idea. Note that they will be executed each time you add the idea.
- **on_remove** is applied similarly to `on_add`, however, it's executed when the idea is removed instead.
- **cost** is the cost in political power to add the idea. If not specified, assumed to be 150 political power.
- **removal_cost** is the cost in political power to remove the idea. It is rarely used.
- **ai_will_do** is the weight of the idea used when AI has to pick an idea. More info can be seen in the [AI module](#).
- **name** assigns the idea's name to use a different localisation key. This can be useful if you plan to modify the name.

Full Idea Example [\[编辑 | 编辑源代码 \]](#)

An example of an idea that uses as many things as possible:

```
XXX_example_idea = {
    ledger = army

    picture = generic_coastal_navy

    cost = 200
    removal_cost = 1000

    available = {
        has_government = fascism
    }

    cancel = {
        has_idea = ZZZ_other_example_idea
    }

    visible = {
        NOT = {
            has_idea = ZZZ_other_example_idea
        }
    }

    allowed = {
        has_war = yes
    }

    allowed_civil_war = {
        always = yes
    }

    on_add = {
        if = {
            limit = {
                NOT = {
                    has_country_flag = chosen_idea
                }
            }
        }
    }
}
```

写法，例子，应有尽有，不会英文？翻译软件。

小结

正如教程开头所说，本文的目标是把读者变成一个遇到问题之后知道去哪找，找什么的人，而不是把网上已有的内容抄下来变成一个不全的百科全书。

不过还存在着一部分概念上的东西 wiki 上不会特别强调但我觉得很重要的，这些将在下一节中讨论。

第八节 和游戏对话

钢四 MOD 的代码是如此贫乏，以至于促成了这一节的出现，你是否发现，绝大部分情况下，你见到的所有代码都是 $XXX = YYY$ 和 $XXX = \{ YYY \}$ 的变种？

$A = 2$ ，意思是把 2 赋给 A，或者说 A 的值是 2。

$set_variable = \{ X = 1 \}$ 又是什么呢？

想象钢铁雄心 4 是一个人，写 MOD 或者说写代码就是与它对话，那么当你写下一行代码，就相当于说：“我要你干一件事。”

接下来，游戏会问：“要干什么事？”

$set_variable$ 这个词，就是在告诉游戏：“我要你设置一个变量。”

自然，游戏接下来的问题是：“设置哪个变量？设置为几？”

为了让游戏明白你接下来的话语是针对“设置变量”（ $set_variable$ ）而说的，你需要在 $set_variable$ 后面加一个“ $=\{\}$ ”，大括号内代表“这里头的话是括号外等于号指定的命令说的补充说明。”

于是你在大括号内添加了“ $X = 1$ ”，代表“设置 X，设置为 1”。

一整句话翻译过来便是：“我要你完成设置变量这个任务，任务分为 X 和 1 两项”。

游戏理解了你的话语，把游戏内名为 X 的变量设为了 1。

交予程序一个任务（指令/函数）和所需的内容（参数），就是钢四 MOD 在干的事情。

你还有第二件事情可干，那就是创造一个东西并给予它属性：

```
ideas = {
    country = {
        ANC_troops_for_ai_1 = {
            picture = ANC_troops_for_AI
            allowed = { always = yes }
            modifier = { training_time_factor = -0.1 }
        }
    }
}
```

将一个内容如上的 txt 文件放入 common/ideas 目录代表什么？首先 common/ideas 也相当于一个命令，代表“你即将听到的东西来自 common/ideas 目录”，这个时候游戏就会做好准备，因为它知道这是用来存放国家精神的目录。

我们分解开看，ideas = {, “我要定义一些国家精神了：”

country = {, “接下来要说的是 country 类型的国家精神：”

ANC_troops_for_ai_1 = {, “一个内部名为 ANC_troops_for_ai_1 的国家精神，它要有以下属性：”

picture =, “图片是：”

ANC_troops_for_AI, “内部名为 ANC_troops_for_AI 的”

allowed = {, “启用条件是：”

always =, “总是为”

yes, “真”（和上面两句话结合呢，“总是启用”）

}, “启用条件说完了。”（极为重要，如果你不说你已经说完了，游戏可不知道什么时候该停下，所以漏括号的后果一般都很严重）

modifier = {, “它拥有以下修正：”

training_time_factor =, “training_time_factor 修正，其值为：”

-0.1, “负零点一。”

}, “修正说完了。”

}, “ANC_troops_for_ai_1 这个国家精神的属性说完了。”

}, “country 类型的国家精神说完了。”

}, “这个文件里的国家精神定义完了。”

程序是很死板的，开启了一个对话，就必须结束它之后才能提其他东西。

有了这种思维，你就很容易发现代码中的错误，我们再来看第六节中的题目：

```
OPTION_A = {  
  allowed = {  
    WhatTriggerIsThis = ?  
  }  
  
  allowed_civil_war = {  
    always = yes  
  }  
  
  removal_cost = -1  
  
  picture = generic_build_infrastructure  
  
  modifier = {  
    production_speed_infrastructure_factor = 0.1  
    production_speed_rail_way_factor = 0.1  
  }  
}
```

allowed = { WhatTriggerIsThis = ? }翻译过来——

“启用条件是：WhatTriggerIsThis 的值为？ 启用条件说完了。”不正确，因为游

戏首先就不知道 WhatTriggerIsThis 是个什么条件，自然无法往下理解。

```
OPTION_D = {  
  allowed = {  
    original_tag = GER  
    always = no  
  }  
  
  allowed_civil_war = {  
    always = yes  
  }  
  
  removal_cost = -1  
  
  picture = generic_build_infrastructure  
  
  modifier = {  
    production_speed_infrastructure_factor = 0.1  
    production_speed_rail_way_factor = 0.1  
  }  
}
```

allowed_civil_war = { always = yes removal_cost = -1

“在内战中启用的条件有这些：总是为真、移除费用为-1”游戏傻眼了，因为他还

等着你给出下一个内战中启用条件，你却说了个其他不相干的玩意，寄。(和机器对话，不但单词要正确，语法也要正确。)

而你要怎么知道 `allowed_civil_war` (或者任何属性/命令/修正) 代表内战中启用的条件呢？看 wiki，或者自己参考原版现有例子悟出来。

小结

钢四代码“有始必有有终”而且“不抱幻想”，布置了一个任务就要有结束语，也不可以出现事先没有约定好的符号。

附录

1.A 游戏目录

正版用户：右键钢铁雄心 4 的游戏图标，在下拉菜单中找到属性



选择本地文件，选择浏览



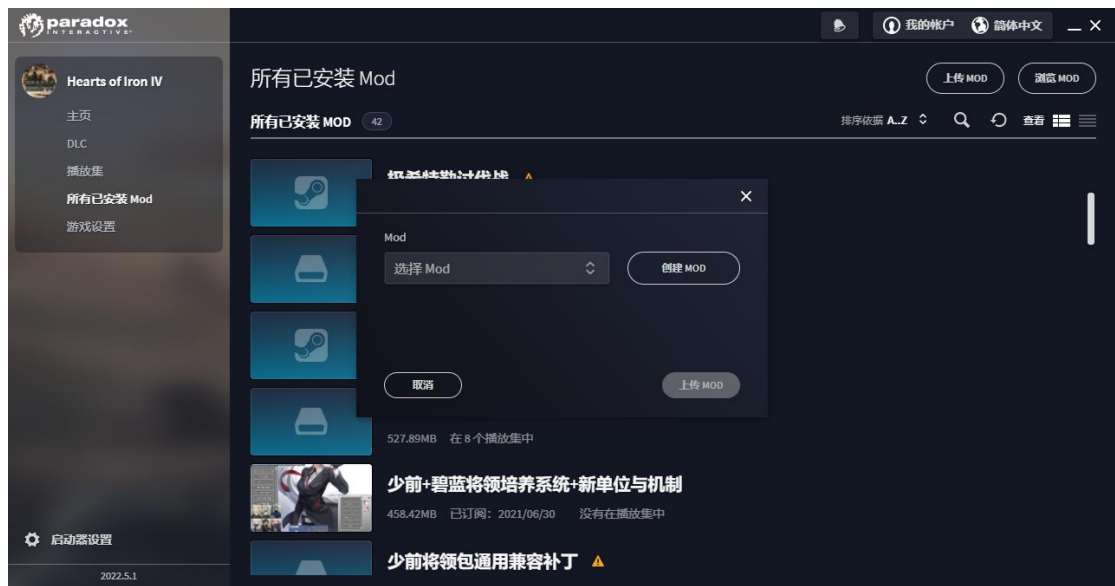
接下来打开的就是游戏目录了。

盗版用户：取决于你的 hoi4.exe 在哪里，它和 dowser.exe 在一起，总之就是那个你启动游戏要用到的 exe 所在的位置。 [返回第一节](#)

1.B 创建 Mod

打开启动器，选择“所有已安装 Mod”，选择“上传 MOD”，点击“创建 MOD”。

（如果提示网络错误请往下多看一段，后面有解决方案）



你会看到这样的界面：

×

名称

版本

目录

mod/

标签

☐ Alternative History

☐ Balance

☐ Events

☐ Fixes

☐ Gameplay

☐ Graphics

☐ Historical

☐ Ideologies

☐ Map

☐ Military

☐ National Focuses

☐ Sound

☐ Technologies

☐ Translation

☐ Utilities

返回

创建 MOD

名称就是 MOD 的名称，可以随便填，比如填“测试 MOD”。

版本也可以随便填，建议填成当前游戏版本（本教程写就时代表填 1.11.10）

目录代表这个 mod 文件的存放位置，前面的“mod/”无法删除可以无视，我们只在乎其后的内容，mod 目录命名规则和操作系统文件夹与文件命名规则一

样，不允许特殊符号，你可以填“Test_Mod”

标签代表如果这个 mod 被上传到创意工坊或 P 社论坛会有哪些标签，其内容具体含义请读者自行翻译，本教程不考虑上传故随便勾选几个就可以。

但天有不测风云，由于某种原因，你可能会卡在点击“上传 Mod”按钮那一步，也就是点击后会出现网络错误提示，无法继续。这个时候，你可以选择开加速器或者等一段时间再试，但我们怎么能被这种小小困难打倒？手动创建！

首先定位到“我的文档\Paradox Interactive\Hearts of Iron IV\mod”目录内。（如果你不知道“目录”是什么意思，文件路径的意思，如果你也不知道“文件路径”是什么意思，请百度或者多用一段时间电脑）

此电脑 > 文档 > Paradox Interactive > Hearts of Iron IV > mod >

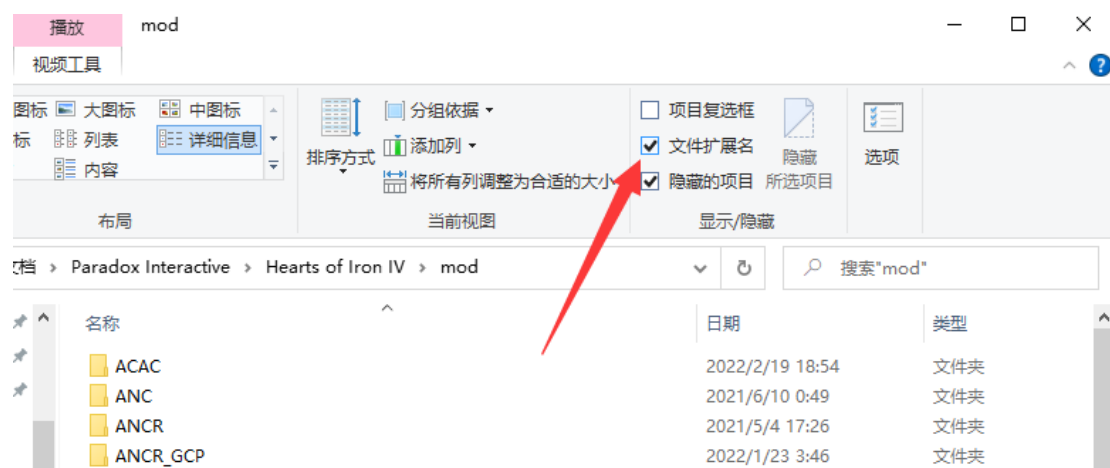
一般来说看起来是这样的↑，由于我的文档文件夹（实为“Documents”）的具体位置因不同电脑而异，我无法给出绝对路径。

上一自然段提到的目录在下文中将被称为“mod 存放目录”。

如果你安装过 mod，mod 存放目录内应该已经有了一些东西，如果你从来没安装过任何 mod，甚至可能都不存在这个文件夹（如果我的文档\Paradox Interactive\Hearts of Iron IV\下没有一个叫 mod 的文件夹，创建一个就可以）

这里需要插播一下钢四载入 Mod 的部分原理，游戏会首先读取一个“引导文件”，这个引导文件会告诉游戏 Mod 的各项属性，比如名称、支持版本号、文件位置等等，我们首先要创建的就是这个“引导文件”。

引导文件统一为.mod 格式，位于 mod 存放目录内，首先创建一个名为 Test_Mod 的文本文档文件——如果你创建后这个文件不叫“Test_Mod.txt”而是“Test_Mod”，你需要先打开拓展名显示（下图为 win10 示例）



将“Test_Mod.txt”修改为“Test_Mod.mod”，并用任意文本编辑器打开（可以是教程一直在用的 VSCode，也可以是其他的，总之能打开 txt 的就行）添加以下内容：

name=

tags={}

supported_version=

path=

其中，“name=”便对应着游戏启动器创建 mod 界面里的名称，相应地，“tags={}”代表标签，“supported_version=”代表版本，“path=”代表 mod 文件所在位置。

那么就很好填了，填完后应该是这样的：

name="测试 Mod"

tags={

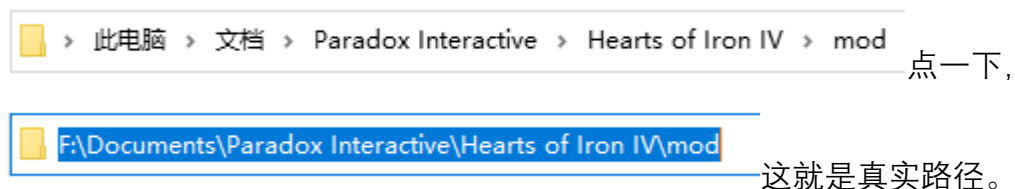
 "Graphics"

}

supported_version="1.11.10"

path="F:/Documents/Paradox Interactive/Hearts of Iron IV/mod/Test_Mod"

特别要注意最后的 path=, 就像我在本节一开始所说, “我的文档”以及后续的 mod 存放目录在不同电脑上的位置可能不一样, 比如我这个电脑就在 F 盘根目录下 (F:/), 你的在哪?



所以我们刚刚创建的这个名为“Test_Mod.mod”的引导文件其内容就会告诉游戏: 有一个名为测试 Mod 的 mod 在 F:/Documents/Paradox Interactive/Hearts of Iron IV/mod/Test_Mod 路径下等待加载。

当然, 下一步就是在 mod 存放目录下创建一个叫 Test_Mod 的文件夹 (你可能要问, 那引导文件给出的路径可不可以不在 mod 存放目录里? 可以, 但你最好别这么干)

这些做完, 再打开启动器时应该就能看到一个叫“测试 Mod”的本地 mod 了。

[返回第一节](#)

4.A 戴着镣铐跳舞

尽管钢铁雄心 4 已经是一个非常“灵活”的游戏了, 但其核心代码仍然不对玩家开放, 故我们所做的一切修改实际上都是开发者“预料到”或者说“允许”的, 而

一部分原版机制的实现过程是不透明的，这一特点你会在教程结尾有更深入的理解，所以知道它管用就够了，有时候不用计较太多。

一句话总结：驴语如此，照做就是

[返回第四节](#)