僵尸毁灭工程MOD教程（更新2022-06-10）

​ MOD制作需要代码、建模、PS多个技能，如果你有以上技能经验会帮助你更好学习和制作MOD。并不是需要你全部都会，要看你制作的MOD是否需要使用以上技能，而且可以与别人合作组成团队。

本教程只包含代码相关教程，可能未来会加入其它。

如无特别说明，教程演示代码和截图只包含原版游戏内容。

# 准备

## 工欲善其事，必先利其器

一个好的开发软件可以让你在开发MOD时事半功倍。

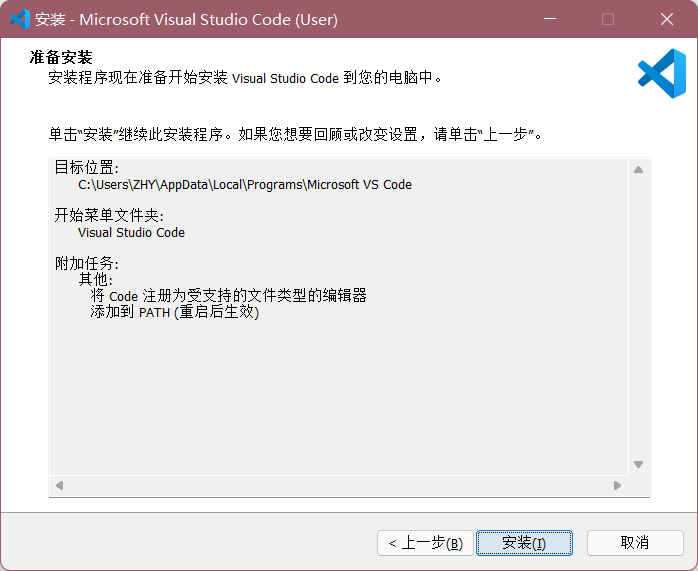
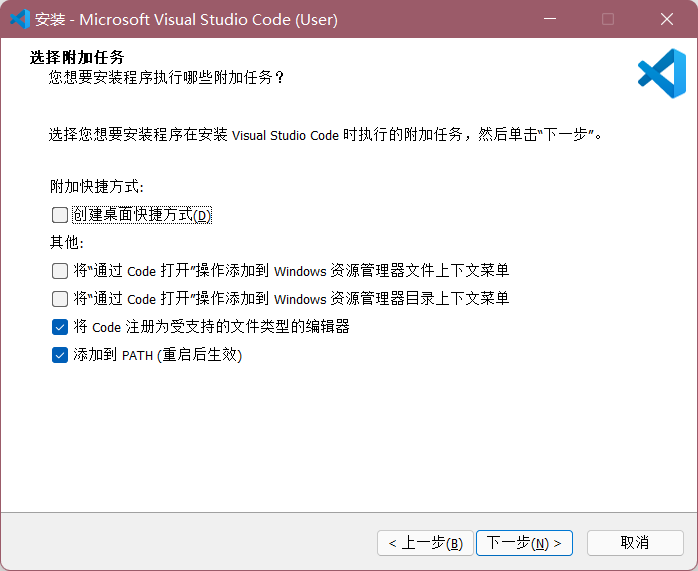
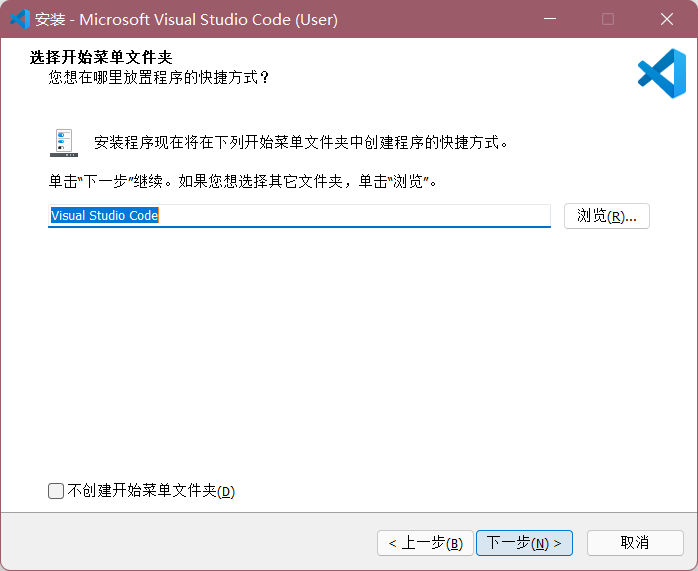
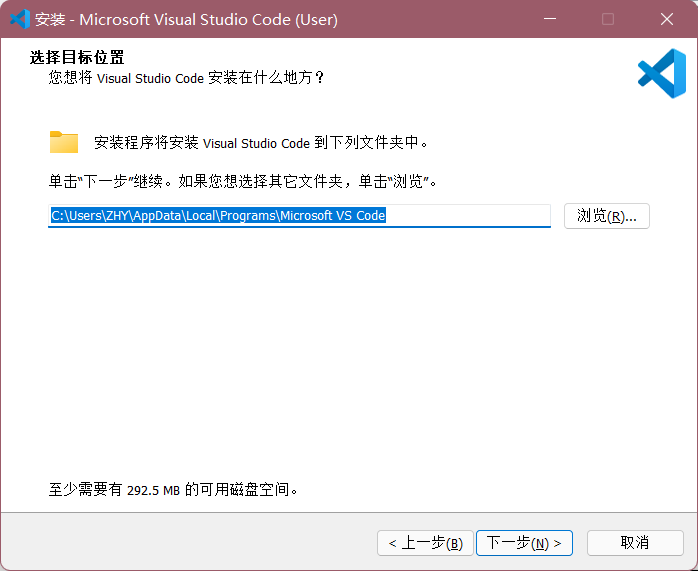
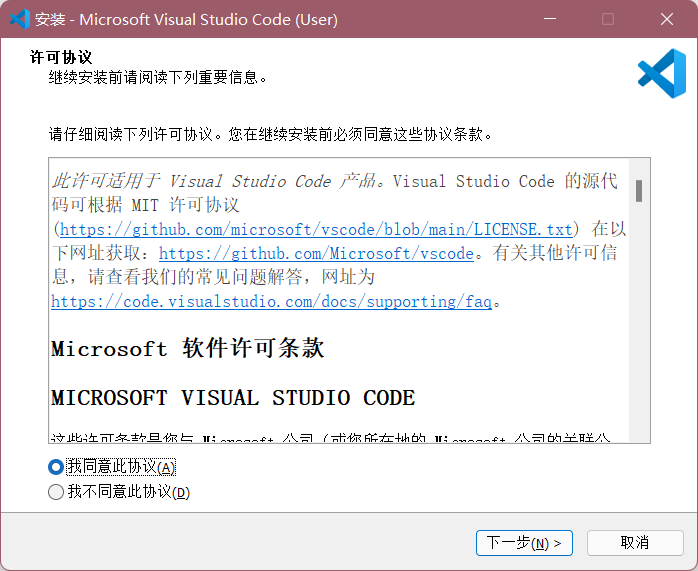
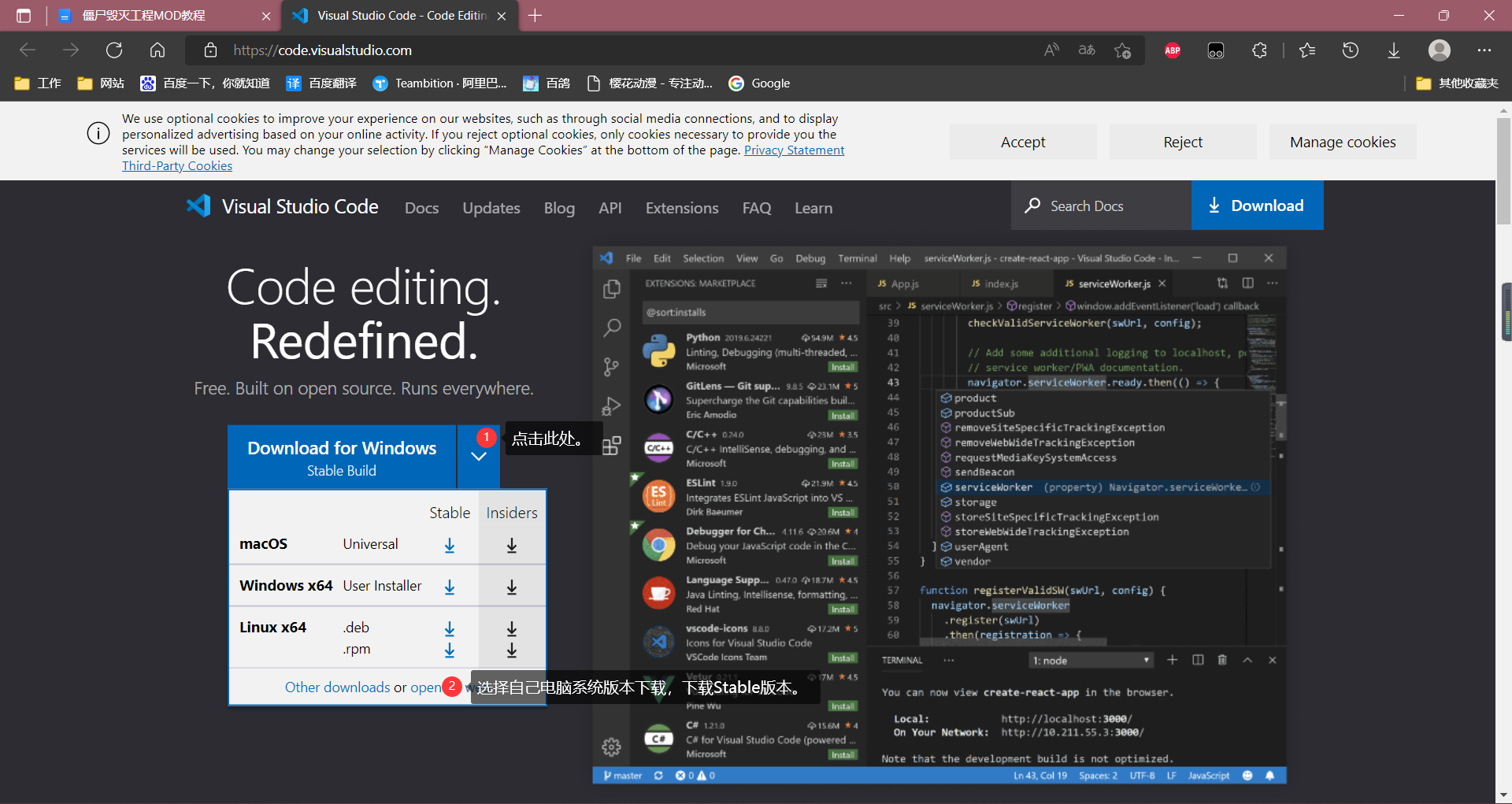
### 代码软件

[Visual Studio Code官网](https://code.visualstudio.com/)

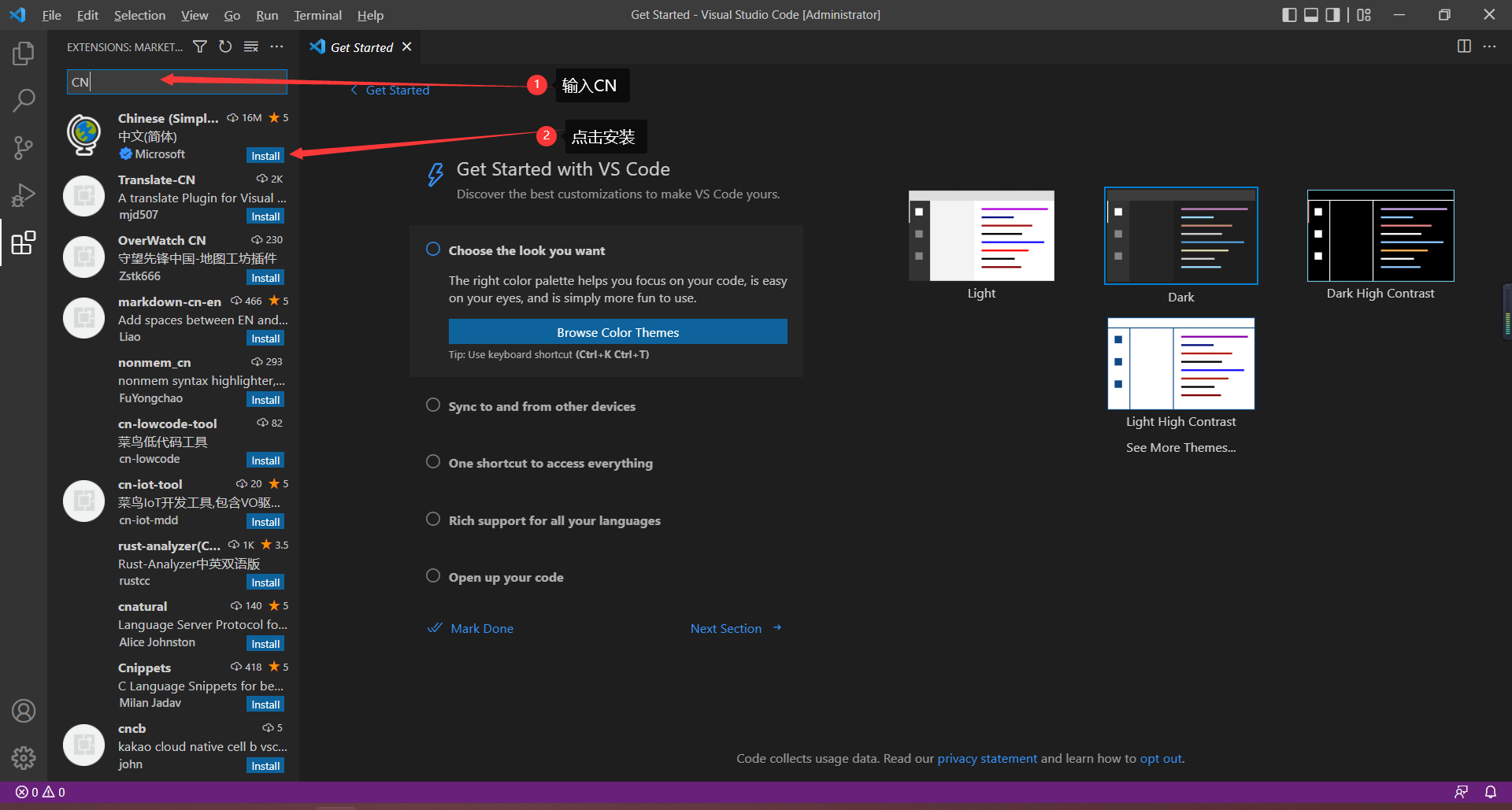
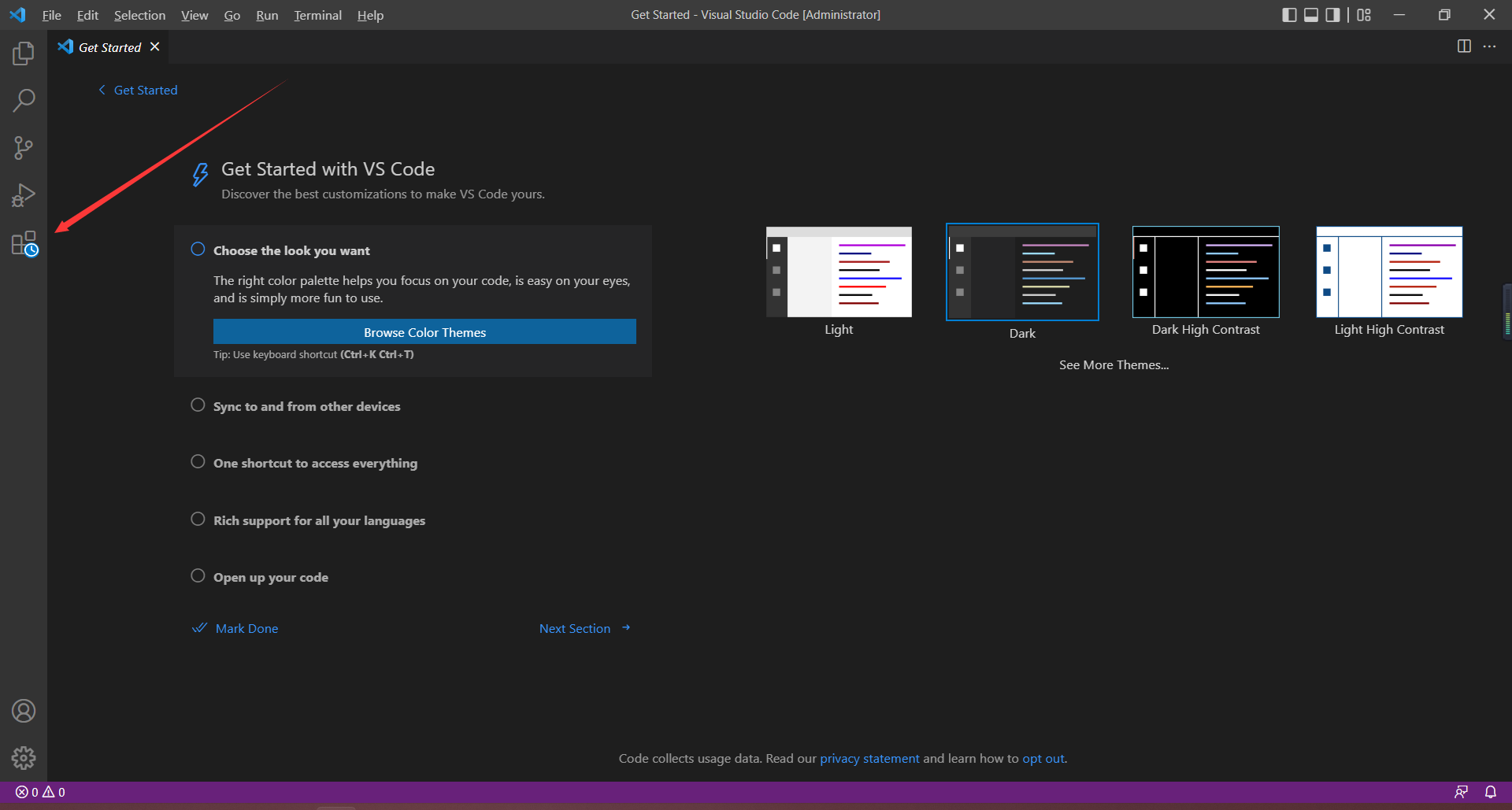
[IntelliJ IDEA官网](https://www.jetbrains.com/idea/)

这里我推荐两款软件Visual Studio Code和IntelliJ IDEA,。PZ的开发组使用的就是IntelliJ IDEA，但需要安装环境和破解，所以该教程推荐新人使用Visual Studio Code，有兴趣自行安装IntelliJ IDEA。

#### 下载和安装



#### 添加中文语言插件和LUA插件

重启软件



#### PZMod插件（后续步骤可跳过，只是简化开发过程）



#### MOD模板生成

使用git工具克隆远程模板库进行开发

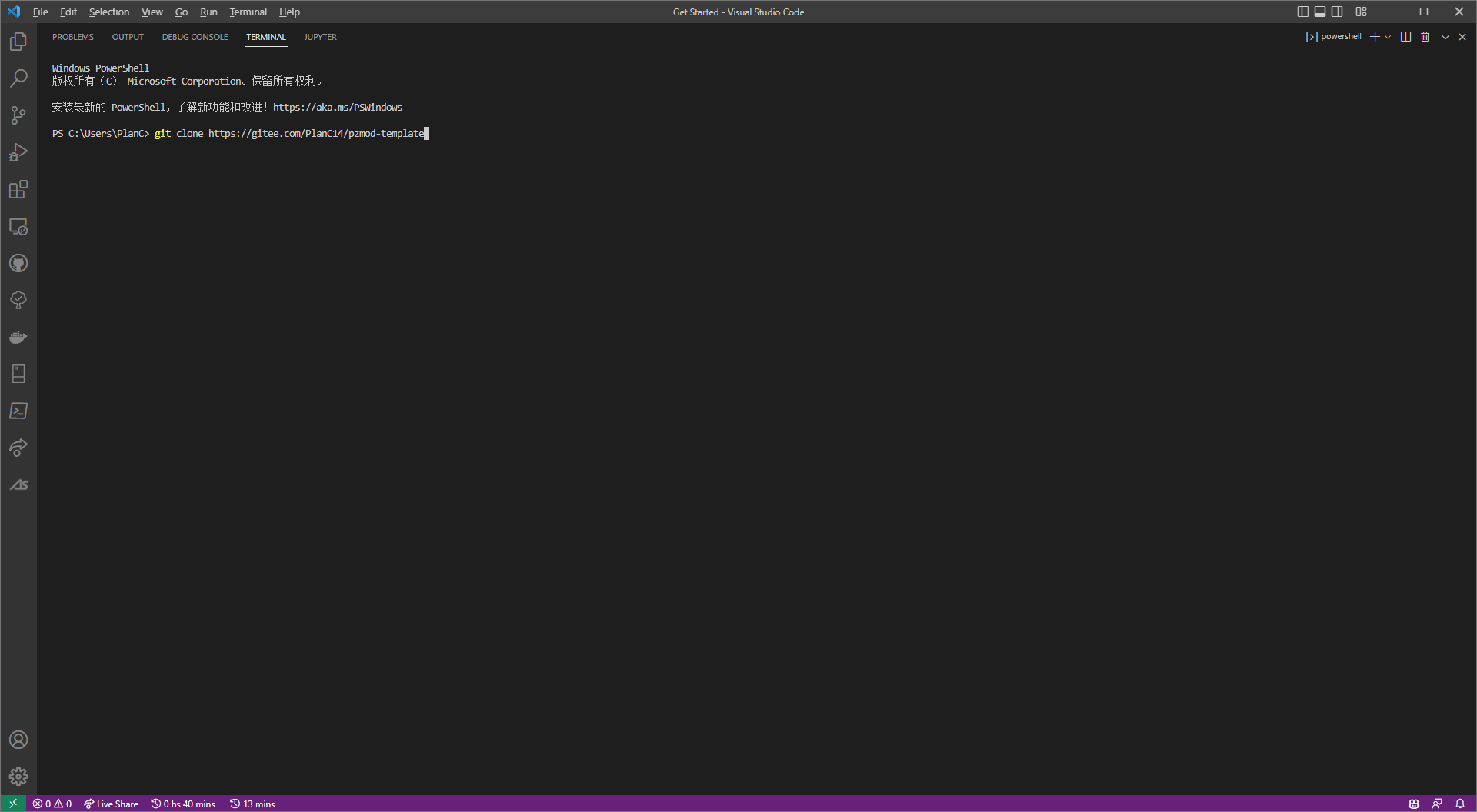
##### 1.1.1.4.1. 下载git工具

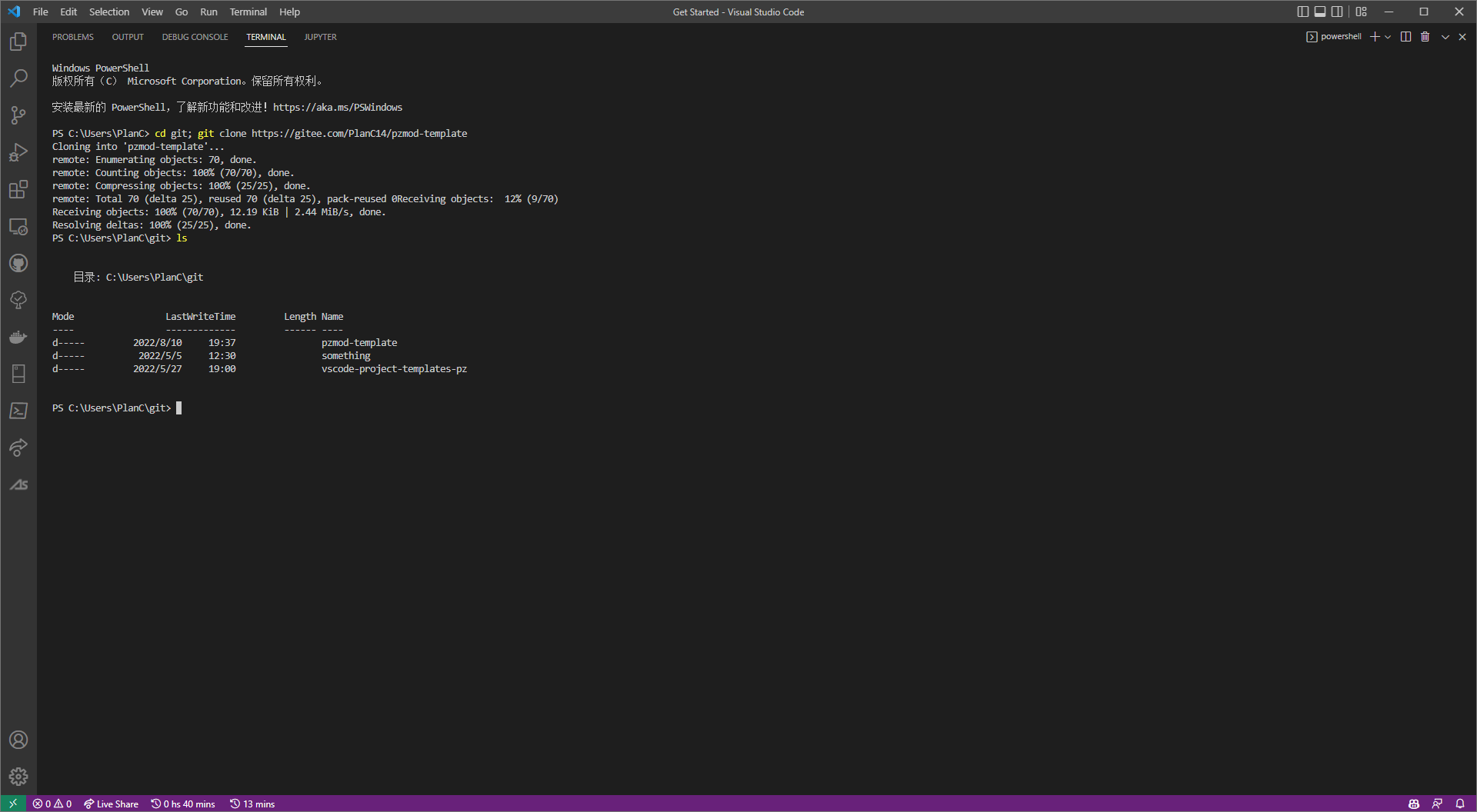
32位下载：<https://registry.npmmirror.com/-/binary/git-for-windows/v2.37.1.windows.1/Git-2.37.1-32-bit.exe>

64位下载：<https://registry.npmmirror.com/-/binary/git-for-windows/v2.37.1.windows.1/Git-2.37.1-64-bit.exe>

##### 1.1.1.4.2. 克隆模板

在vscode中可直接使用Ctrl+Shift+`打开内置控制台进行克隆模板





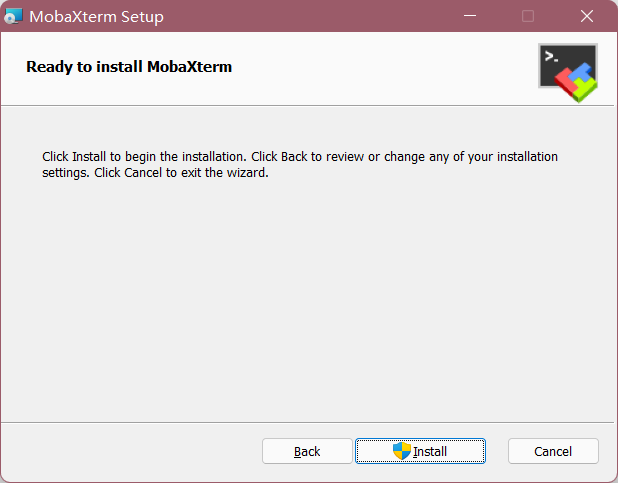
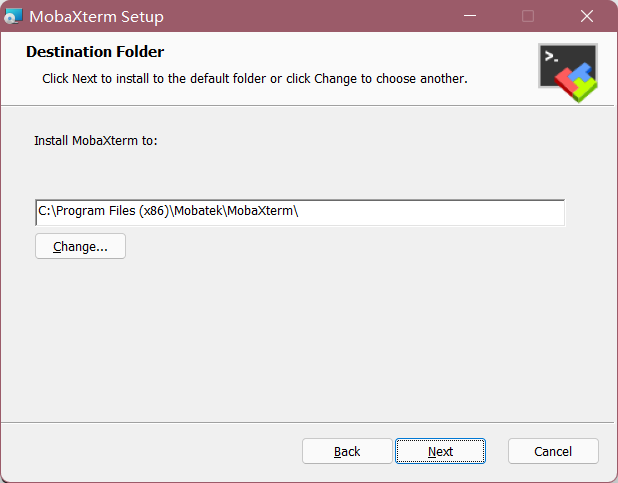
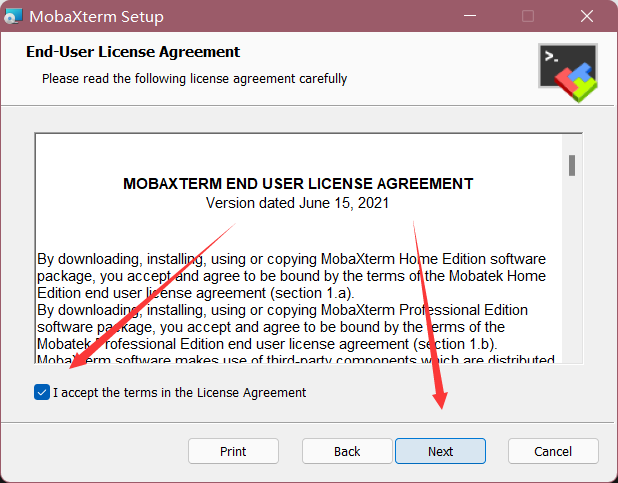
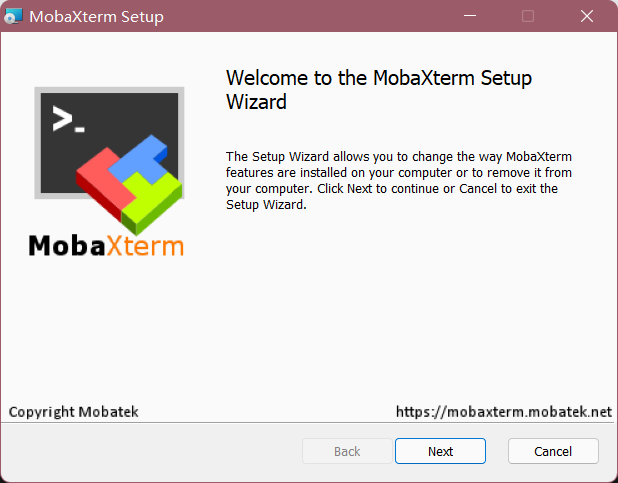
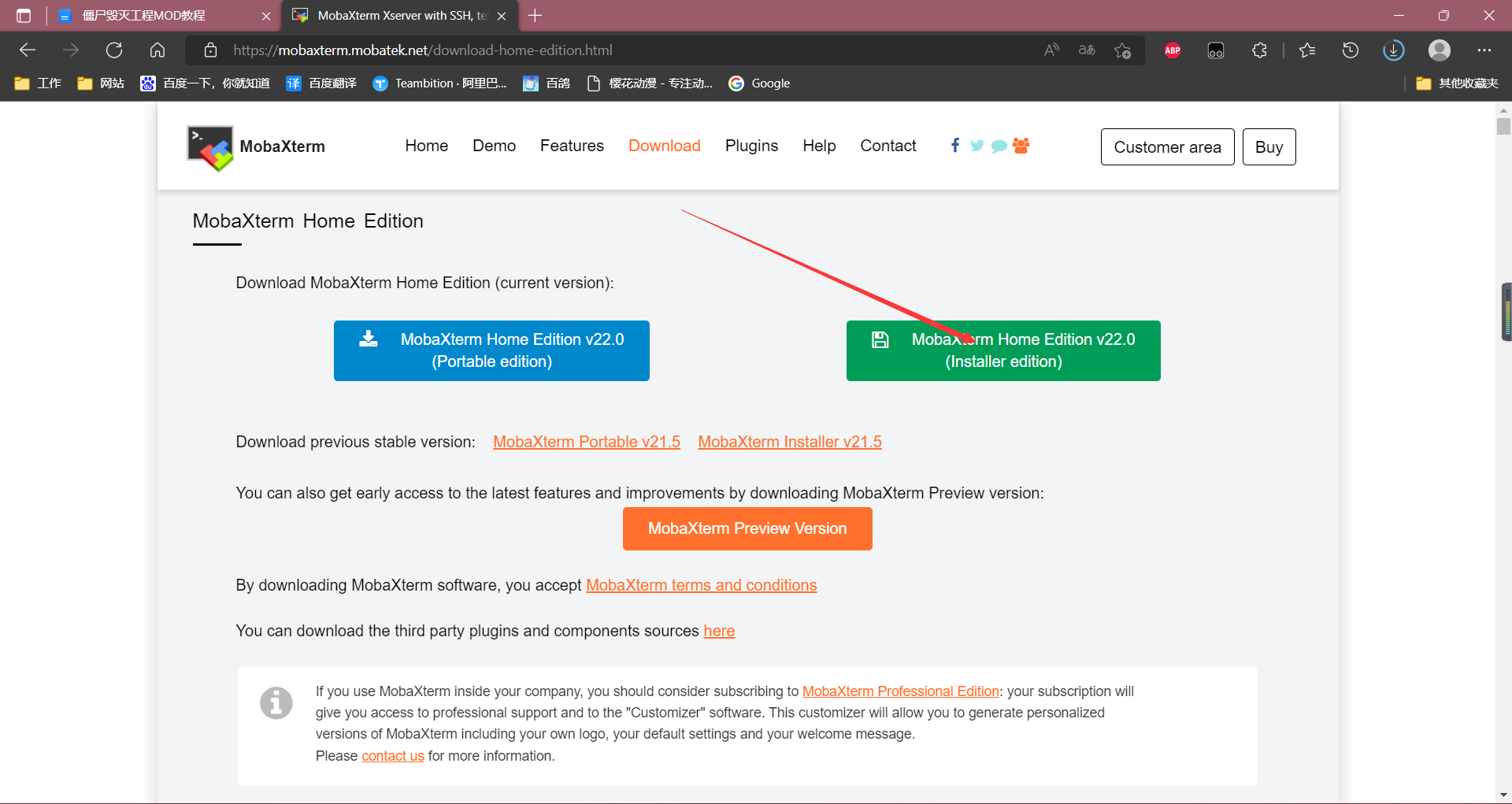
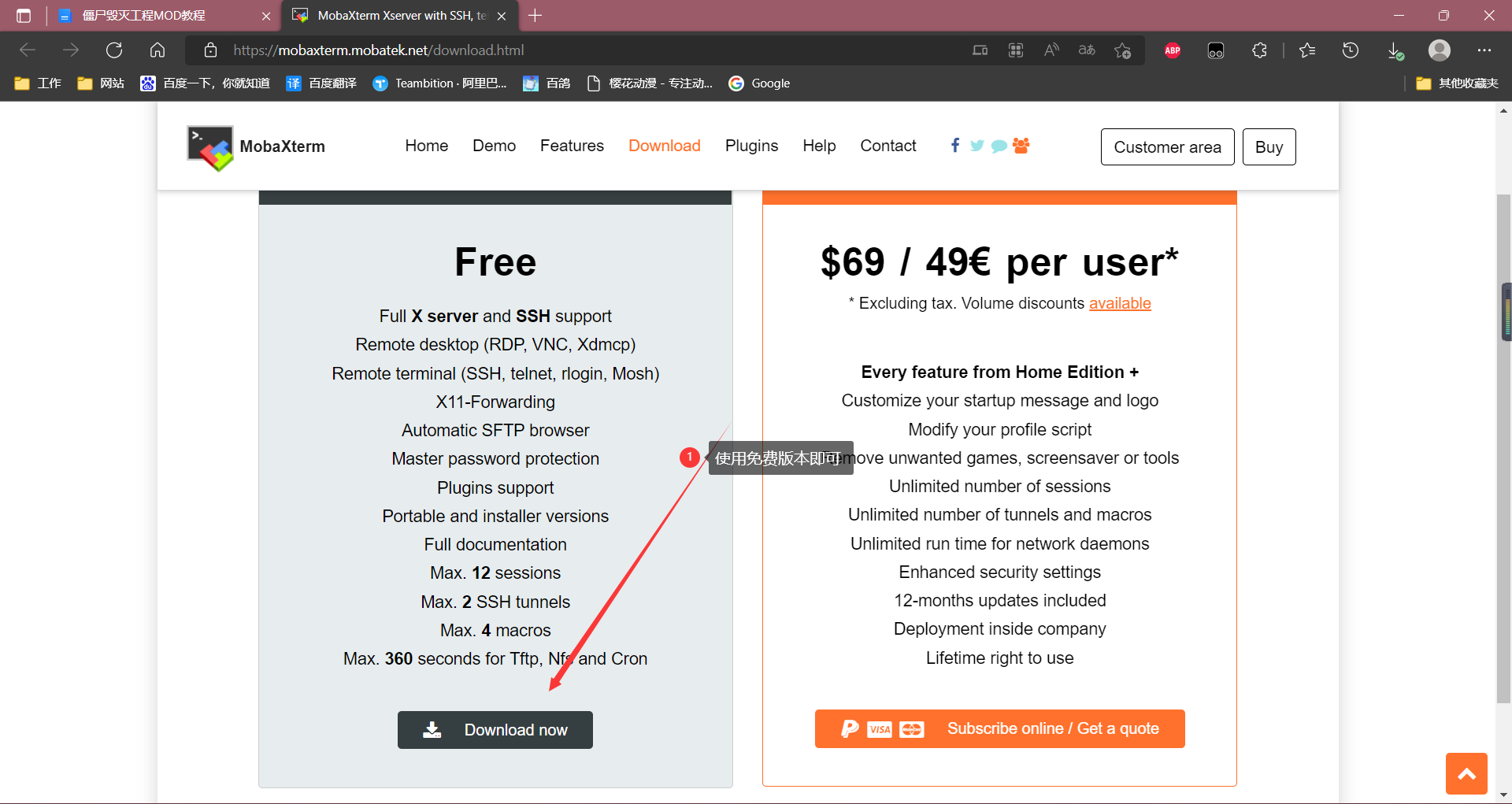
克隆完成后即可打开相应的文件夹进行mod开发，更详细的git说明请参考5.2章节

### 日志查看软件

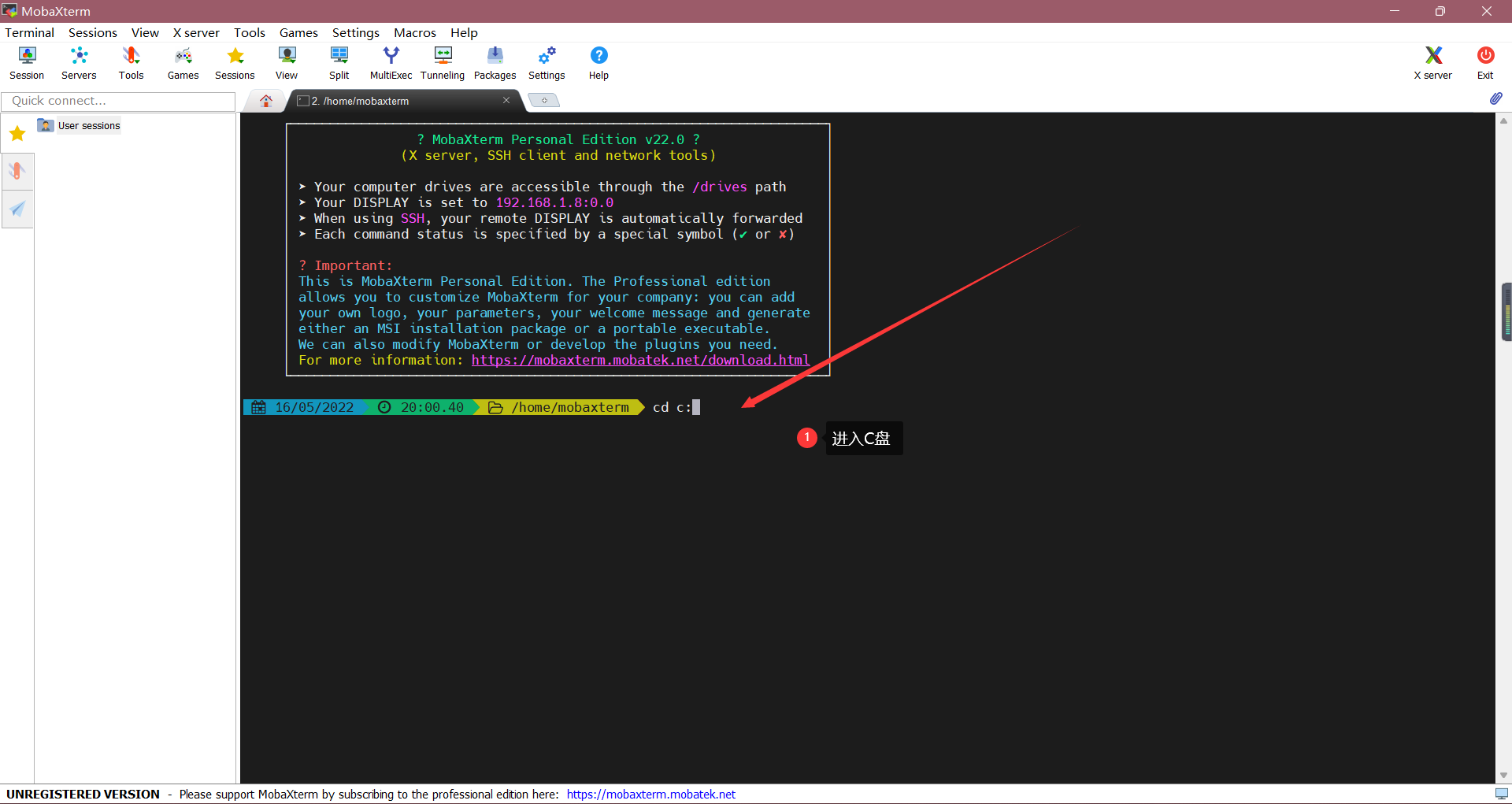
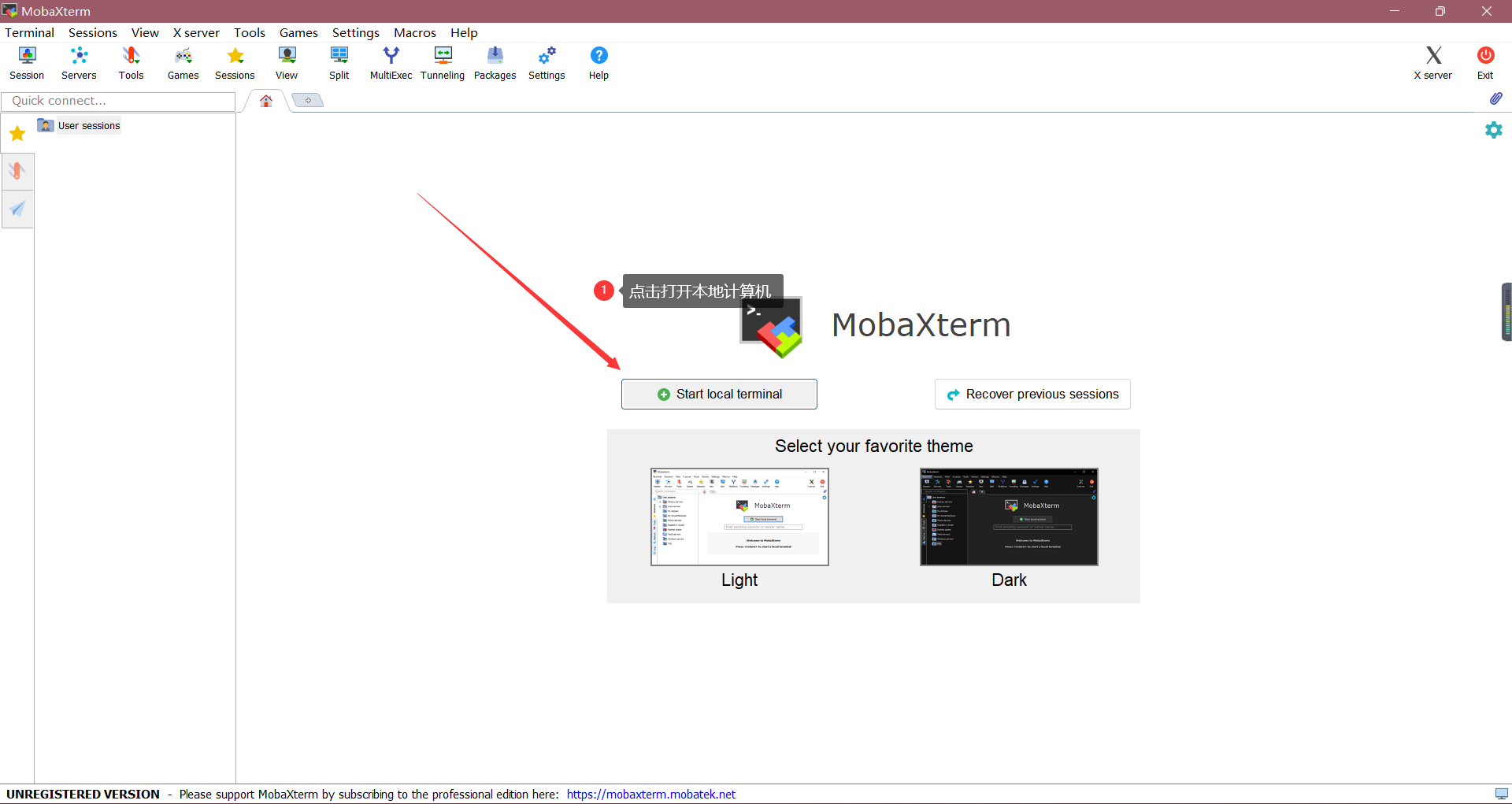
[MobaXterm官网](https://mobaxterm.mobatek.net/download.html)

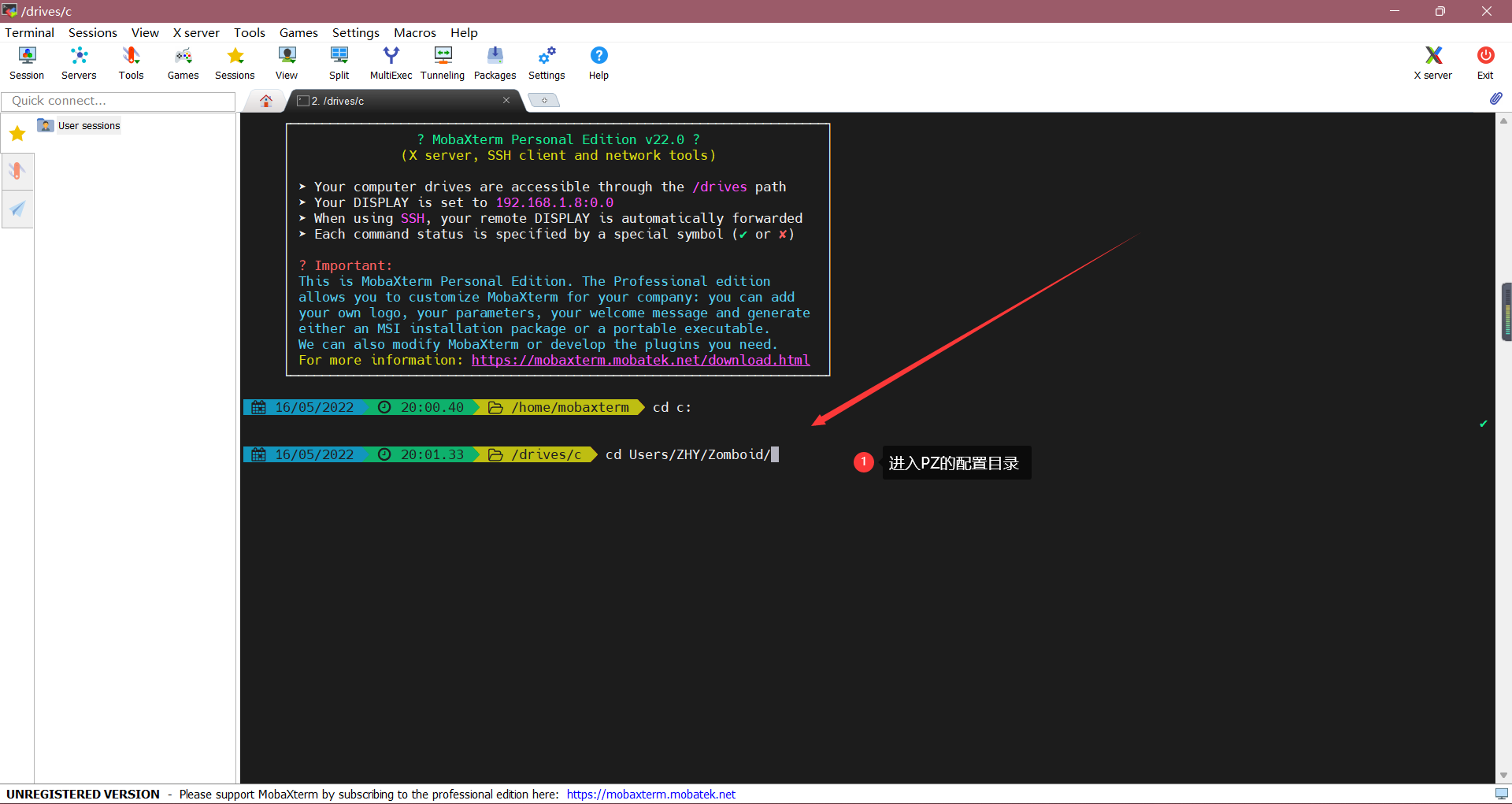
游戏自带日志窗口小而且不能多开、搜索，记事本软件无法实时滚动。MobaXterm是一款连接服务器的命令工具，它允许我们操控服务器一样操作本地计算机。

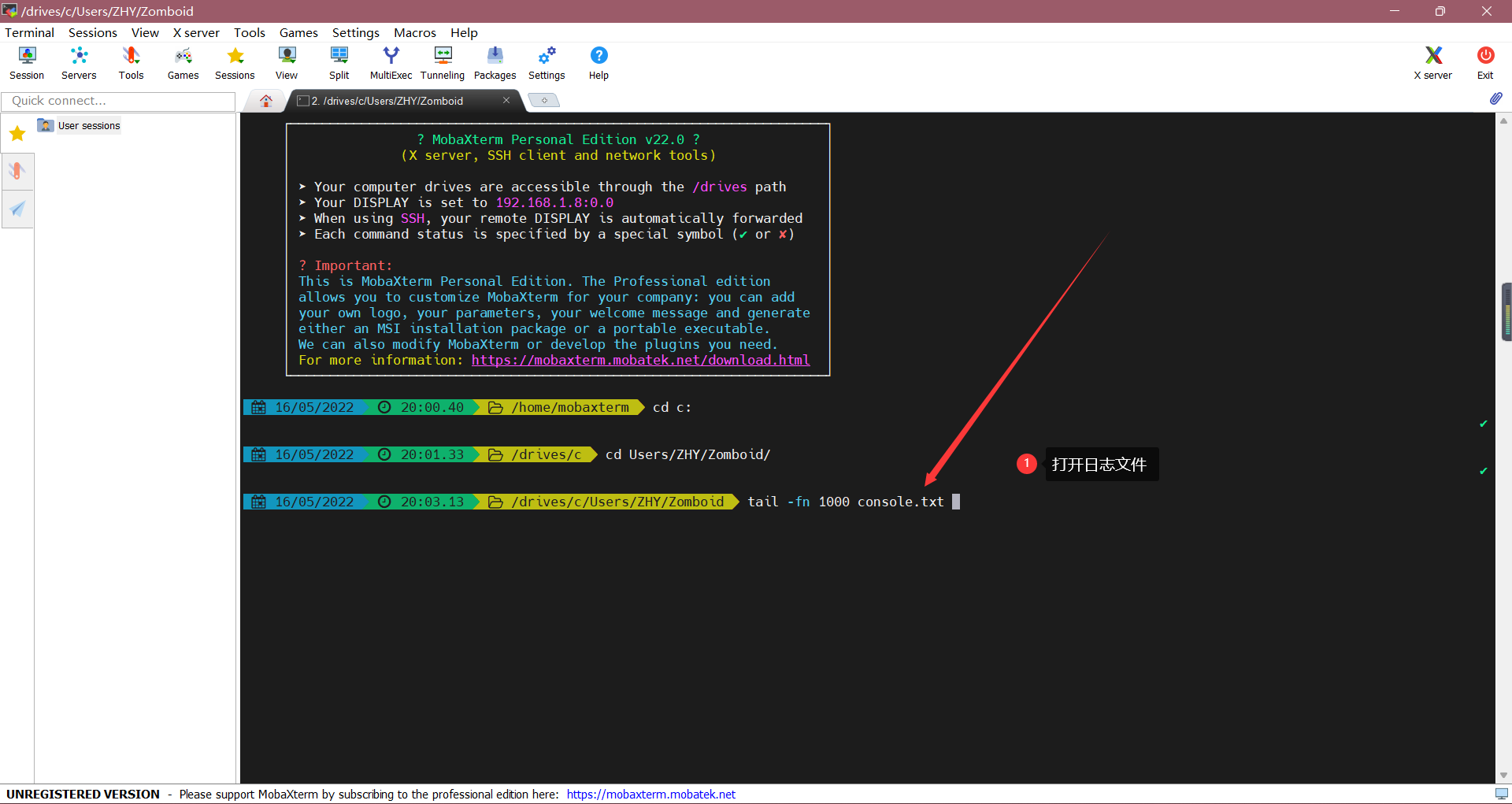
#### 下载和安装

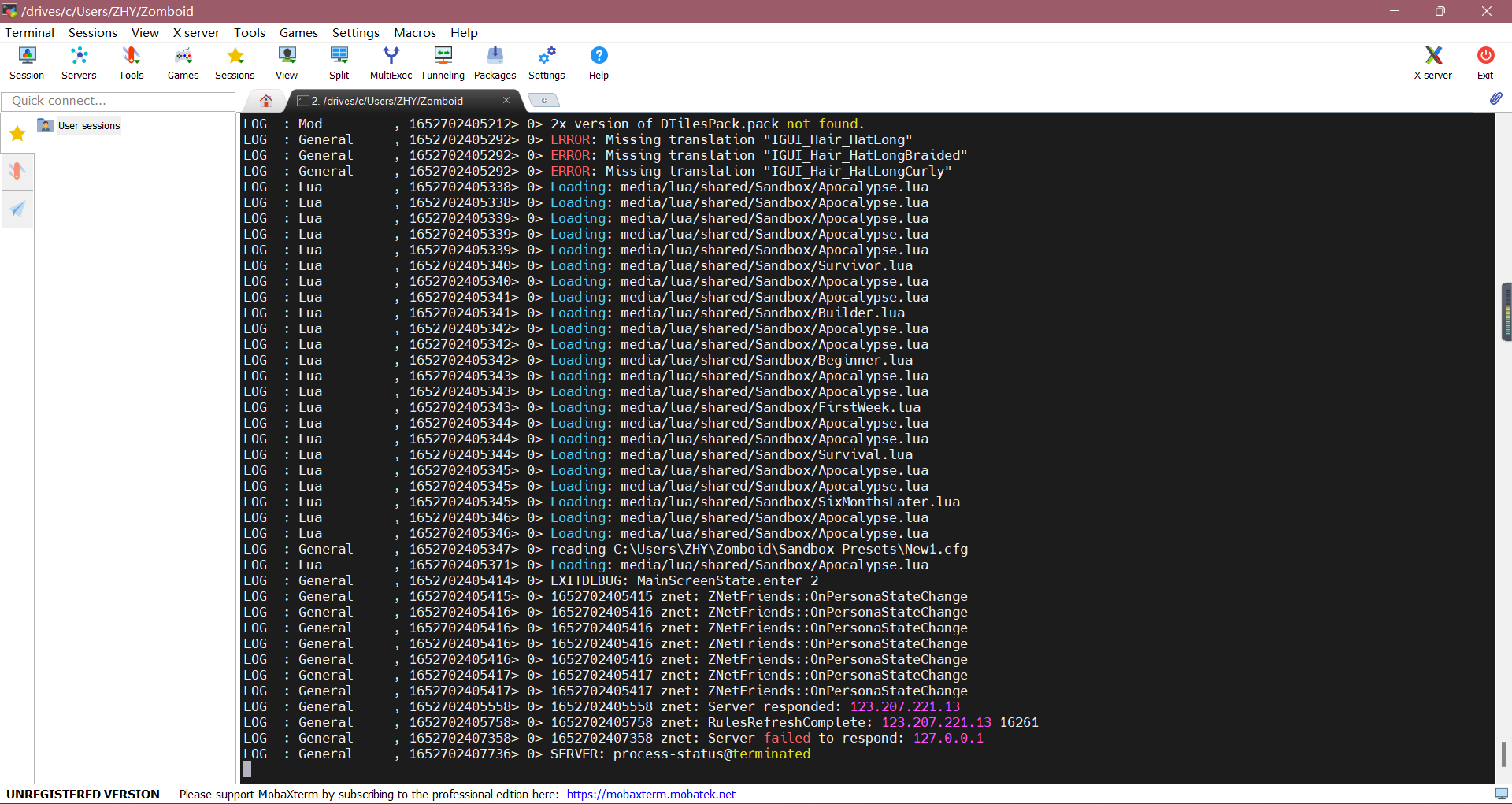


#### 查看日志文件





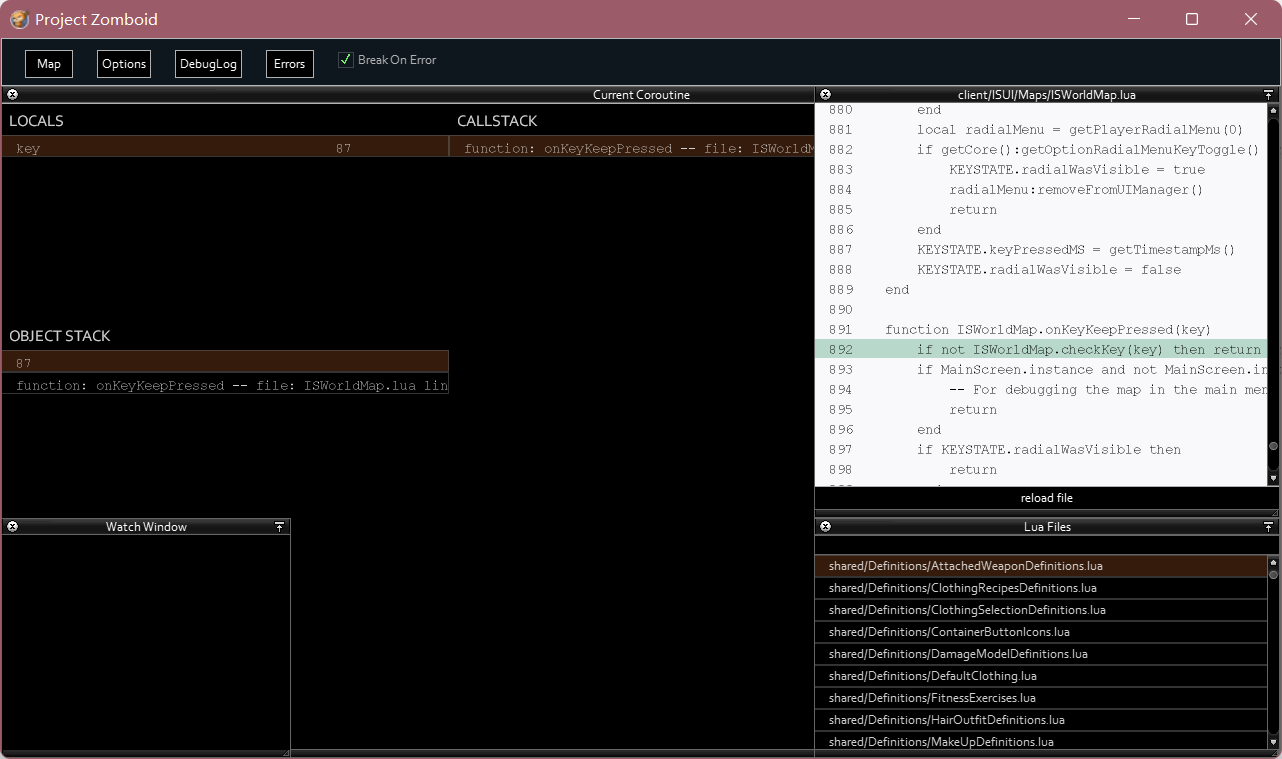
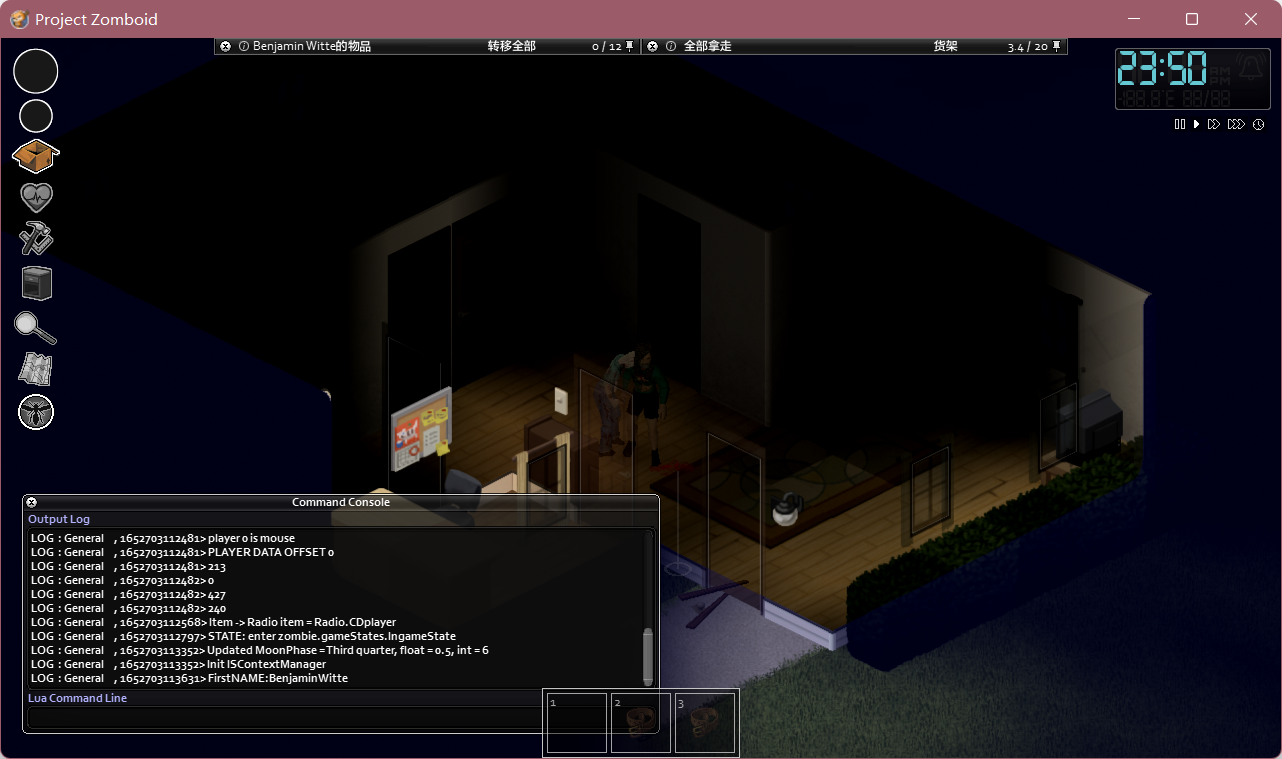




## 调试模式



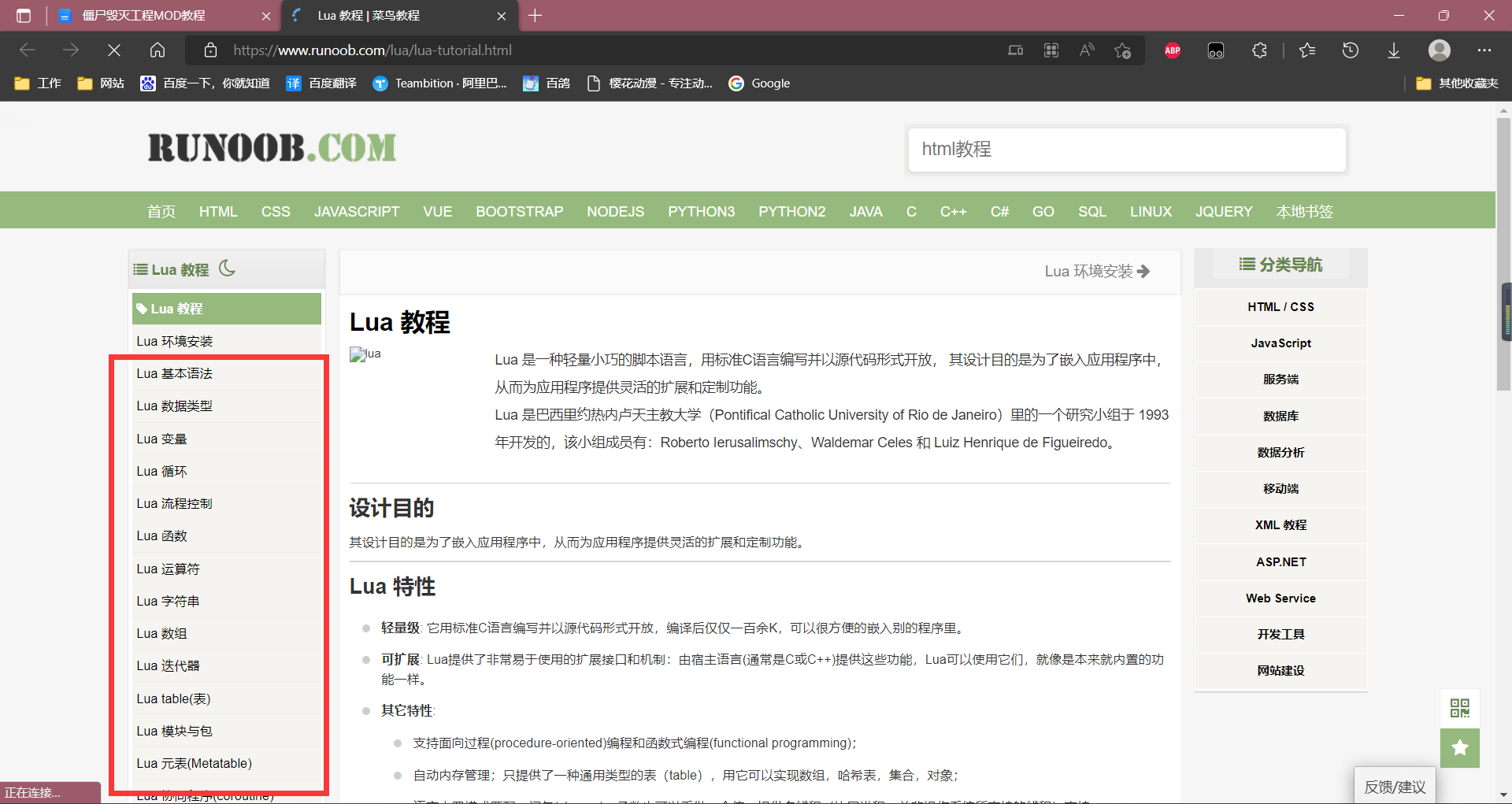
 输入-debug再打开游戏会进入调试模式，会自带作弊功能（不习惯英文也可以订阅调试菜单MOD），当lua代码出错时会弹出到报错页面。

 报错页面也可以通过按F11打开关闭，可以看到出错的代码位置和相关变量值。在右下角还能重新加载lua文件，方便快速更改代码查看效果。

## LUA基础

[LUA教程](https://www.runoob.com/lua/lua-tutorial.html)

PZ使用JAVA和LUA开发，但做MOD只需要用到LUA。不需要精通但因大概了解LUA语法。开始学习前最好先阅读红框中内容。

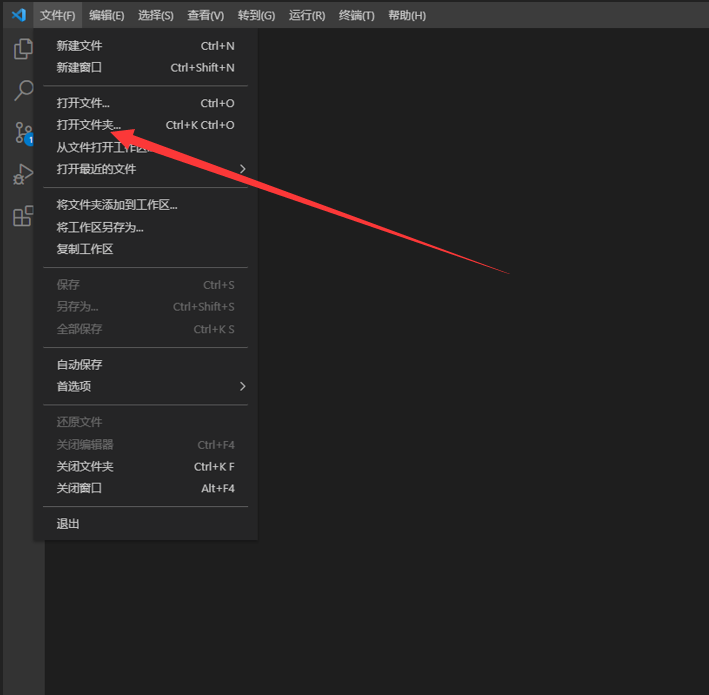


# 基础

## 新建MOD

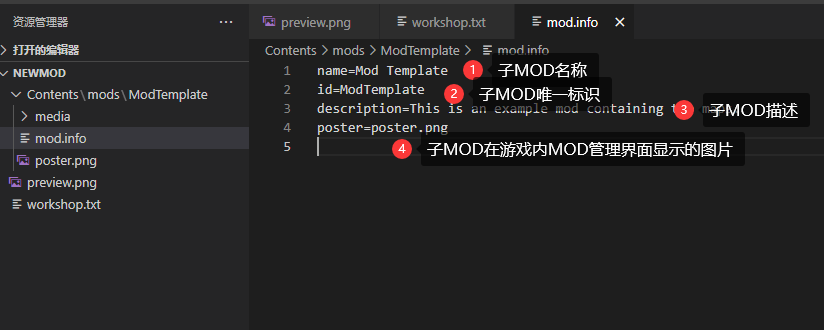
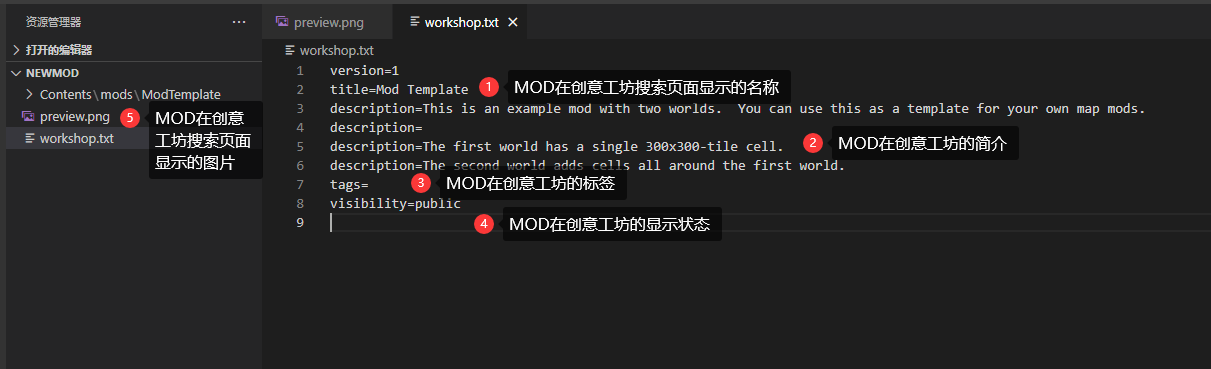
打开文件管理器C:\Users\你的电脑用户名\Zomboid，这个目录是PZ保存存档、配置等等的目录，我们开发MOD就在Workshop下开发。

 打开Workshop文件夹，可以看到ModTemplate。这是官方提供的MOD模板，我们复制一份命名为NewMod。然后用Visual Studio Code打开NewMod文件夹。

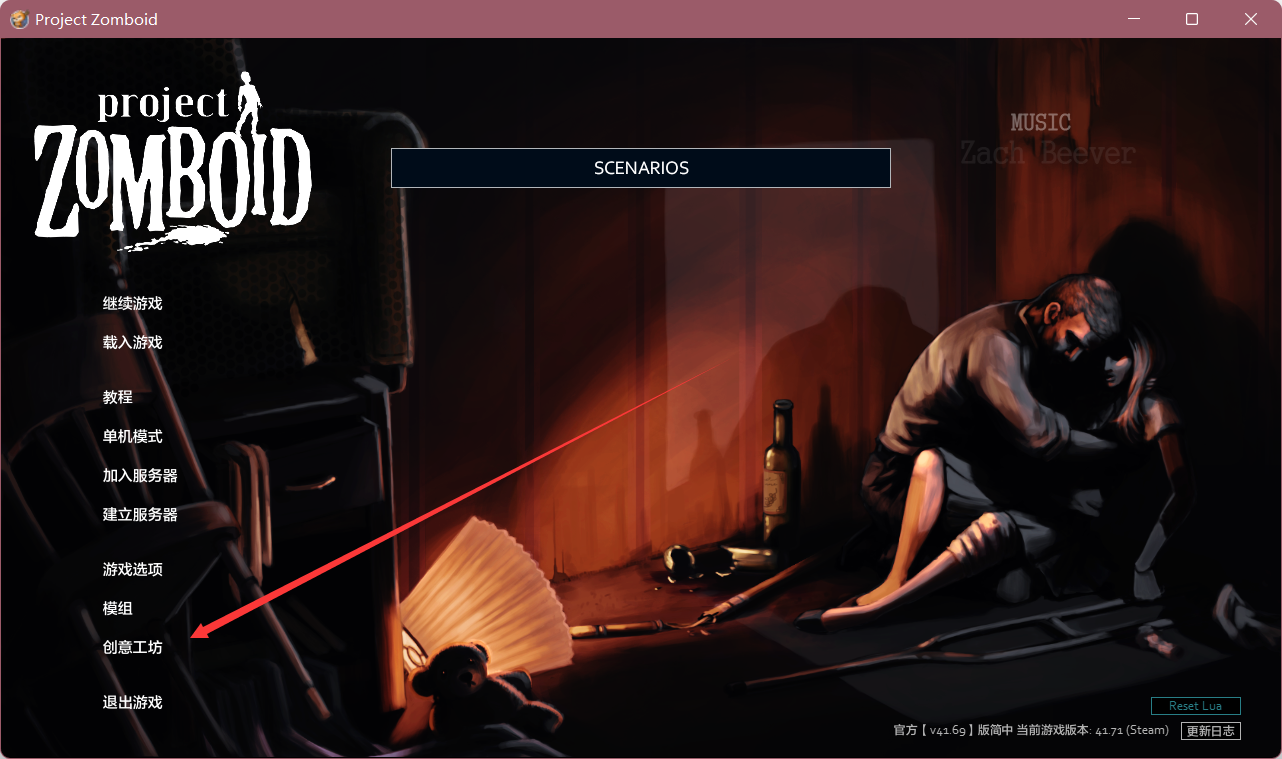


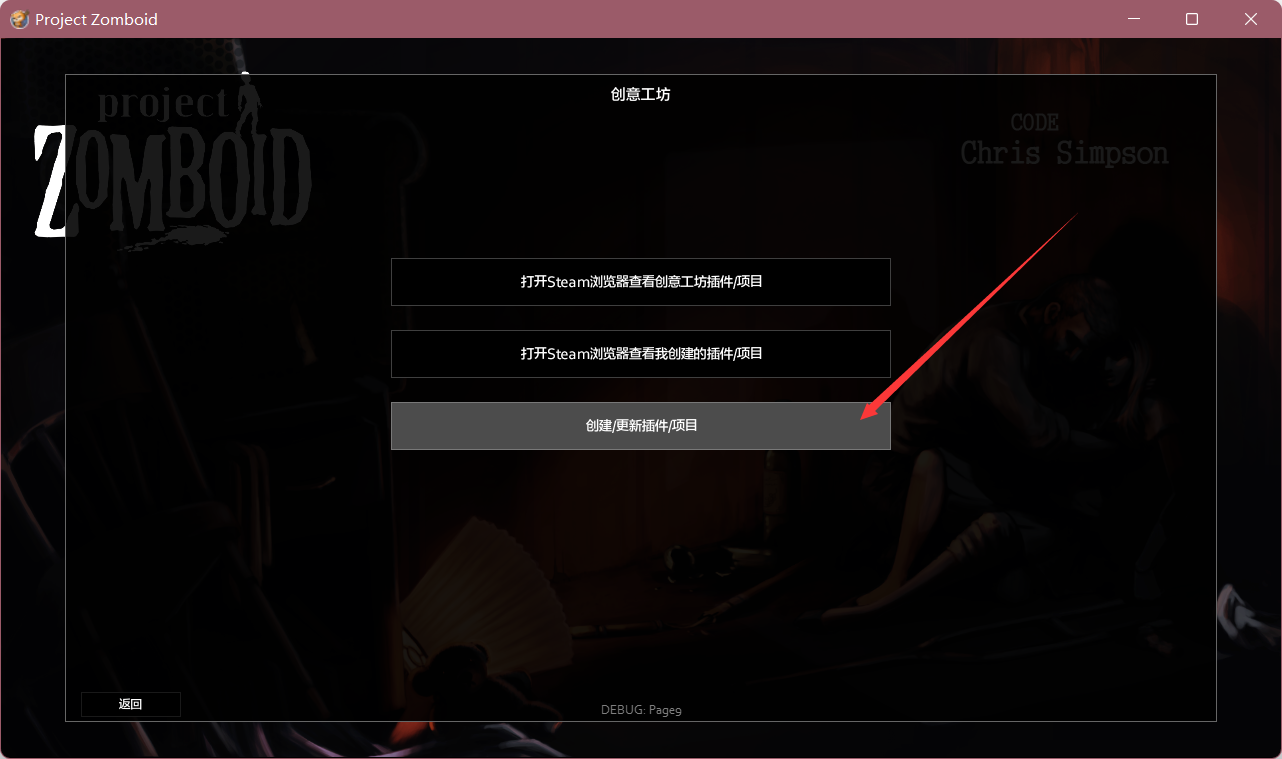
## MOD结构

media下的内容根据你制作MOD需要，自由添加。下面列出常见目录

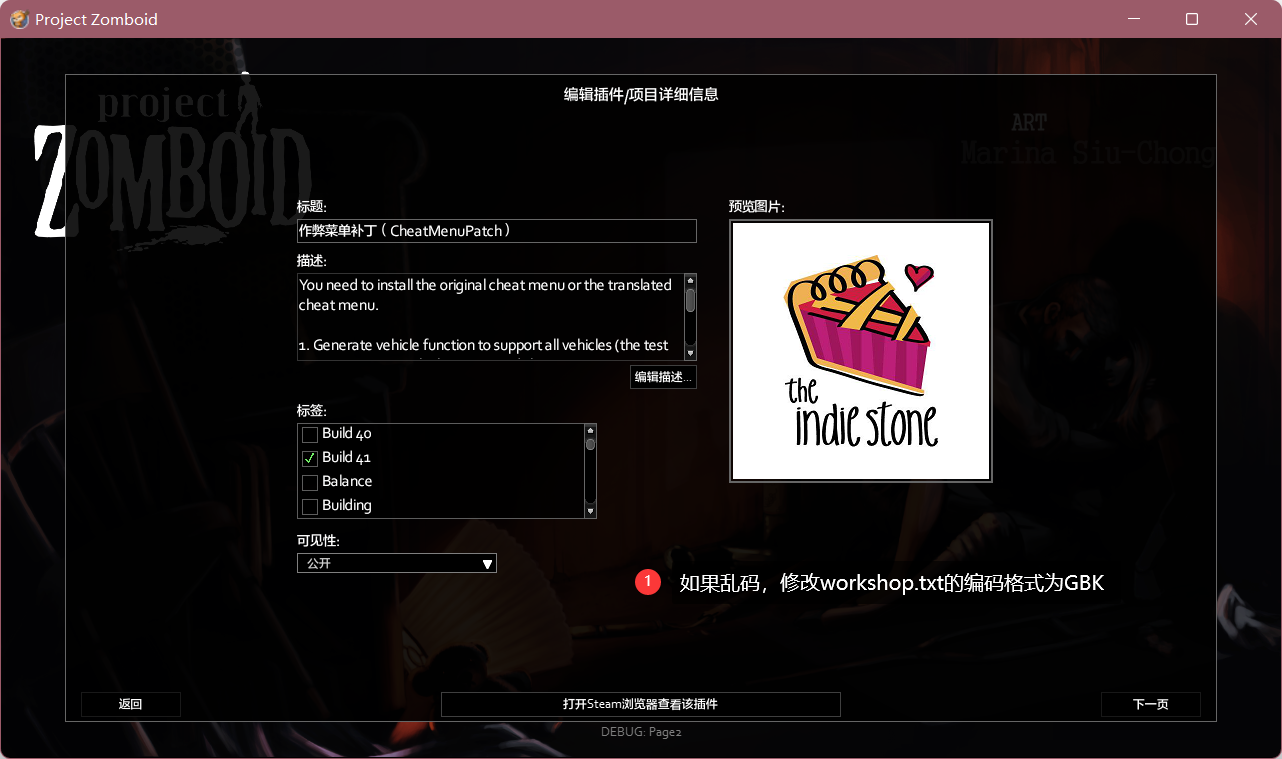


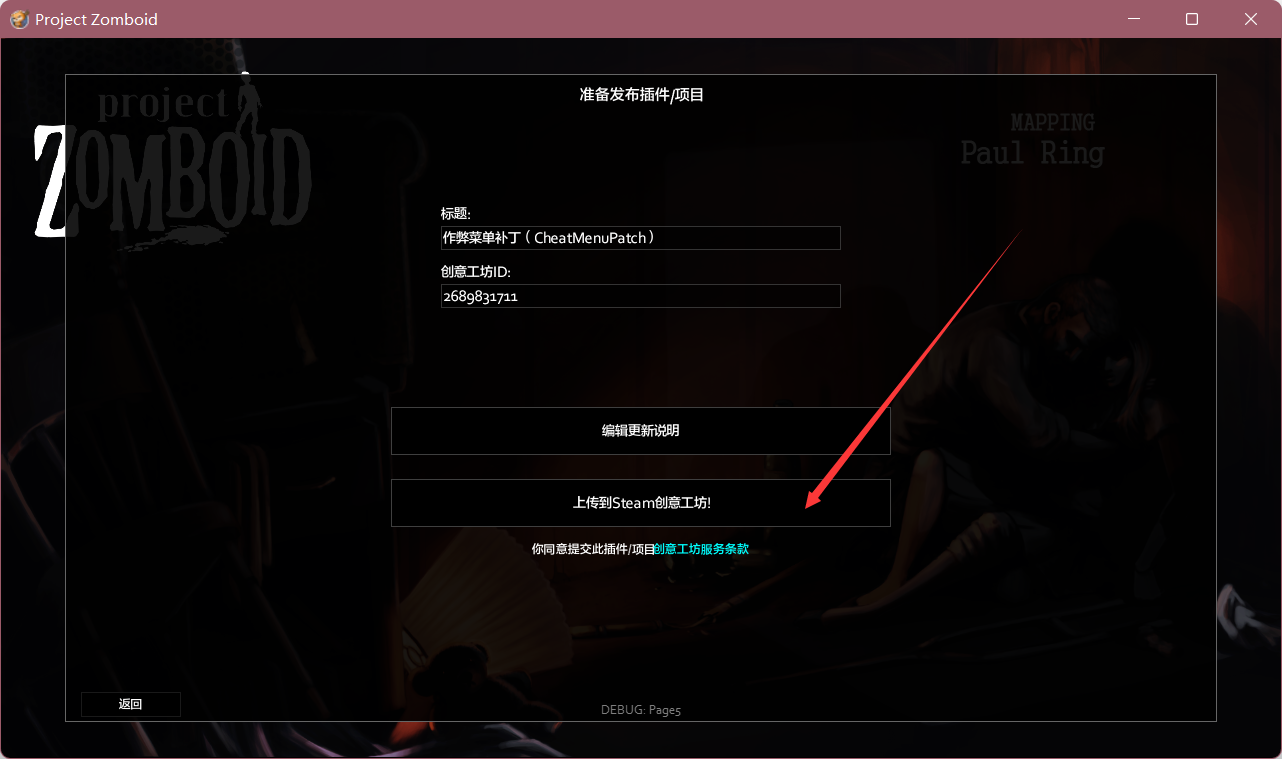
## 上传MOD











## 代码规范

代码规范没有统一标准，适合自己才是最好的，只要做到从一而终。我这列出自己的一些规范，供大家参考。

1. mod.info中的id添加自己昵称缩写，避免与其它MOD重复——昵称\_MOD名。例：QNW\_TaskSystem
2. 物品、配方、函数、变量、文件都尽量取有意义的名字，禁止abc123之类的。
3. 取名使用英文、拼音、中文都可，但不要混用。
4. 英文名和拼音名采取驼峰命名法。例：getName、huoDeMingZi
5. 函数参数名前加下划线以表示这是函数参数。例：
6. lua不使用;做结束符号。
7. lua函数、变量尽量定义本地变量和专有命名空间，详细看4.2。
8. 良好的缩进格式。
9. 自己晦涩或逻辑复杂的地方增加注释。
10. lua中使用字符串，使用getText()的方式。
11. 文件夹使用英文，不使用空格和标点符号。

# 初级

## 物品脚本

物品脚本是位于media/scripts目录下的符合格式的txt文件。

### 注释块

位于/\*\*和\*\*/之间的内容会被游戏忽略，用于备注相关功能。

### module块

物品块必须位于module中。游戏原版大多数物品module是Base。相同的module和item会使物品定义互相覆盖，如果你的目的不是覆盖修改已有物品，推荐使用自定义module，名称最好与MOD名称相同或相关，避免与其它MOD产生冲突互相覆盖。

### item块

物品块包含对物品的定义，物品属性数量众多，这里就不展开详细描述。如果你想知道其它属性用法和含义，可参照游戏原版物品块和MOD的物品块。教程完成后抽空整理属性列表

[物品属性列表](https://docs.qq.com/sheet/DYlRia0FldENoQXNM)

## 配方脚本

配方脚本是位于media/scripts目录下的符合格式的txt文件。

### 注释块

位于/\*\*和\*\*/之间的内容会被游戏忽略，用于备注相关功能。

### imports块

对于配方块，您需要引用其他模块中的项目。通常，当某个项存在于不同的脚本模块中时，必须使用全名（如Base.ScrewsBox）来引用它。通过imports块，你可以省略对应模块名，只需使用ScrewsBox。

### module块

配方块必须位于module中。游戏原版大多数配方module是Base。相同的module和recipe会使配方定义互相覆盖，如果你的目的不是覆盖修改已有配方，推荐使用自定义module，名称最好与MOD名称相同或相关，避免与其它MOD产生冲突互相覆盖。

### recipe块

配方块包含对配方的定义，配方属性数量众多，这里就不展开详细描述。如果你想知道其它属性用法和含义，可参照游戏原版配方块和MOD的配方块。教程完成后抽空整理属性列表

[配方属性列表](https://docs.qq.com/sheet/DYkZ3UE9kYmR6dFhU)

## 修复配方脚本

修复配方脚本是位于media/scripts目录下的符合格式的txt文件。

### 注释块

位于/\*\*和\*\*/之间的内容会被游戏忽略，用于备注相关功能。

### imports块

对于修复配方块，您需要引用其他模块中的项目。通常，当某个项存在于不同的脚本模块中时，必须使用全名（如Base.ScrewsBox）来引用它。通过imports块，你可以省略对应模块名，只需使用ScrewsBox。

### module块

修复配方块必须位于module中。游戏原版大多数修复配方module是Base。相同的module和fixing会使修复配方定义互相覆盖，如果你的目的不是覆盖修改已有修复配方，推荐使用自定义module，名称最好与MOD名称相同或相关，避免与其它MOD产生冲突互相覆盖。

### fixing块

修复配方块包含对修复配方的定义。

|  |  |
| --- | --- |
| Require | 配方适用的物品，多个物品使用;隔开 |
| GlobalItem | 需要的物品，每一个Fixer在原有条件上加上GlobalItem |
| ConditionModifier | 修复效果乘数，默认1 |
| Fixer | 可以存在多个，修复需要的条件，使用;隔开，第一个是需要的物品，后面是需要满足的技能 |

## 进化配方脚本

修复配方脚本是位于media/scripts目录下的符合格式的txt文件。

### 注释块

位于/\*\*和\*\*/之间的内容会被游戏忽略，用于备注相关功能。

### imports块

对于进化配方块，您需要引用其他模块中的项目。通常，当某个项存在于不同的脚本模块中时，必须使用全名（如Base.ScrewsBox）来引用它。通过imports块，你可以省略对应模块名，只需使用ScrewsBox。

### module块

进化配方块必须位于module中。游戏原版大多数进化配方module是Base。相同的module和evolvedrecipe会使进化配方定义互相覆盖，如果你的目的不是覆盖修改已有进化配方，推荐使用自定义module，名称最好与MOD名称相同或相关，避免与其它MOD产生冲突互相覆盖。

### evolvedrecipe块

进化配方块包含对进化配方的定义。

|  |  |
| --- | --- |
| BaseItem | 基础物品 |
| MaxItems | 最多能添加几次材料 |
| ResultItem | 结果物品 |
| Cookable | 是否可以煮熟 |
| Name | 配方显示名称 |
| CanAddSpicesEmpty | 是否可以添加调味品 |
| AddIngredientIfCooked | 煮熟后是否可以添加材料 |
| AddIngredientSound | 添加材料时的音效 |

## 车辆脚本（未完成）

## 地图脚本（未完成）

## 衣服脚本（未完成）

## 国际化

你可能已经发现在上面定义物品、配方等时配置的显示名称只有一种语言，那么怎么实现国际化？让不同语言玩家能看到自己语言显示的物品？

以物品名来说，原版消防斧有一条属性DisplayName = Axe,。但我们在游戏中看到的是消防斧而不是Axe。游戏并不会直接显示Axe而是会去media/lua/shared/Translate下查找对应语言的翻译文件。我们选择中文，游戏就会去media/lua/shared/Translate/CN/ItemName\_CN.txt文件中去查找ItemName\_Base.Axe = "消防斧"。所以当我们添加物品、配方后需要添加相应语言的翻译文件就能实现国际化。下面是一些做MOD时常用的翻译文件。

|  |  |
| --- | --- |
| ItemName\_CN.txt | 物品翻译 |
| Recipes\_CN.txt | 配方翻译 |
| IG\_UI\_CN.txt | UI翻译 |
| ContextMenu\_CN.txt | 菜单翻译 |
| Sandbox\_CN.txt | 沙盒设置翻译 |

在后面内容中还会细讲其它用法。

# 高级

请先完成1.3的内容，下面内容默认你已有一定的lua知识。

## LUA执行顺序

单机

shared->server->client

联机

服务端

shared->server

客户端

shared->client

## 新建LUA文件

在shared、server、client下新建lua文件，lua中的代码就会在游戏启用mod后执行，为了避免污染lua元命名空间和避免与游戏原版、其它MOD互相覆盖冲突。定义函数、变量尽量使用local。如果函数与变量需要跨文件使用，请在自定义命名空间定义，不要在元命名空间定义。自定义命名保证唯一，没有强制要求。推荐我的命名方式——网名缩写\_MOD名缩写\_文件名。

例：我在任务系统（TaskSystem）中新建的ClientCore.lua定义变量和函数

如果你想在多个文件使用相同的命名空间。声明时使用or避免相互覆盖。

如果你需要使用其它文件的变量和函数，请确保在其它文件之后执行，参见3.1LUA执行顺序。如果文件在相同文件夹下使用require来确保在指定文件后执行。

下面示例ClientCore.lua将在ClientCommand.lua后执行。

## 全局函数

游戏将许多JAVA函数注册到LUA的元命名空间方便我们使用，我们可以称其全局函数。

[全局函数列表](https://docs.qq.com/sheet/DYmthdmxsVHFocnN0)

## 事件

游戏自带事件机制，当在一些游戏特殊节点会触发相应事件。

[事件列表](https://docs.qq.com/sheet/DYkR0WmtCRnltV1B3)

### 添加事件回调

添加键盘按下事件的回调，当按下按键时会在日志文件输出你按下哪一个键。

### 删除事件回调

当按下T键时会移除事件回调，再按下按键就不会执行test函数。

### 触发事件

在代码里触发事件，你会发现你没有按T键。

### 自定义事件

通过游戏原有键盘按下事件实现自己的移动事件，当按WASD移动时出发并知道当前前进方向。

## ISBaseObject

推荐先阅读：

[lua 中 . 和 : 的区别](https://blog.csdn.net/baidu_35080512/article/details/85857739)

[lua元表](https://www.runoob.com/lua/lua-metatables.html)

想一个问题，假设你实现了动物系统。

你怎么表示不同动物？不同动物有不同跑的移动速度你怎么维护？甚至同种动物因为成长、衰老、受伤移动速度会改变，改怎么保存不同动物的速度？

你可能尝试这么做，但你会发现dog和cat在run里都会输出15，因为dog和cat本质是一样的都是Animal,你需要使用其它方式保存各自的速度并且自行维护速度的正确性。但这样增加复杂性。我们可以通过元表的方式来实现。

当我们调用new函数时，我们新建一个table——o，并将Animal设置成o的元表再返回o。这样我们每次调用new获得的都是新的table，所以他们speed不会互相覆盖，又因为他们元表是Animal，我们依然能使用Animal的函数run。

在游戏中，已经提前创建好了一个表ISBaseObject。

你会发现我们的new一模一样，并且提供了两个新的函数derive和initialise、新的变量Type。derive是派生函数用于在现有表上派生新的表，它的功能其实和new一样，只是在意义（使用方法）是有区别，new是在当前表新建新的对象，例如从动物新建动物1、动物2、动物3。derive是在当前表派生新的表，例如从动物派生狗、猫、鱼。游戏中lua很多功能都使用到了ISBaseObject，就以UI相关表举例。

ISButton（按钮）是ISPanel（面板）的派生表。

ISPanel（面板）是ISUIElement（UI元素）的派生表。

ISUIElement（UI元素）是ISBaseObject的派生表。

简单来说derive用于扩展表功能，new用于新建同功能表。现在我们对我们的动物系统改造一下。

我们动物表派生自ISBaseObject，新增了奔跑函数。猫狗表派生自动物表，分别新增了抓老鼠和游泳函数。而且因为派生自动物表，猫狗依然有奔跑的函数。

## 玩家

获取玩家对象是经常需要使用的操作。

## 菜单

### ISContextMenu

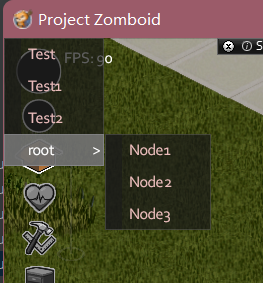
游戏中菜单使用ISContextMenu实现。

#### 创建菜单



#### 新增子菜单（点击事件）

#### 新增嵌套子菜单



### 物品菜单

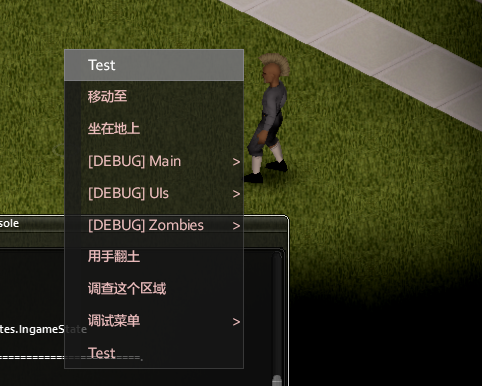
当你在库存面板物品上右键时会触发OnFillInventoryObjectContextMenu事件和OnPreFillInventoryObjectContextMenu事件。



\_items不是一个简单的table，结构比较复杂，我推荐使用游戏自带的ISInventoryPane.getActualItems函数将其转化为一个包含所有选中物品的table。如果你对原\_items结构感兴趣，可以自行调试查看。

### 世界菜单

当你在地面上右键时会触发OnFillWorldObjectContextMenu事件和OnPreFillWorldObjectContextMenu事件。



## ModData

### ModData

游戏中很多Java对象都有getModData()函数，用于获取一个与之相关联的table，用于保存数据，且moddata的数据会在游戏保存时自动保存。

常见含有和使用getModData()函数的对象有。物品、玩家、僵尸、NPC、时间、单元格等等。注意：moddata保存的数据不是联机共享的，比如在你客户端上玩家对象保存的数据在其它玩家客户端和服务端是获取不到的。

如果你要保存一份数据，但这份数据不是单独某个对象的数据。比如统计服务器里所有人一共击杀多少僵尸，你不能保存到玩家对象上，因为玩家在下线状态你是获取不到玩家对象的，一般方案是将数据保存到游戏时间对象上，因为这个对象一直存在且全局唯一。

### GlobalModData

为了解决modData的不足，游戏提供了GlobalModData。

|  |  |  |
| --- | --- | --- |
| 函数 | 说明 | 返回 |
| ModData.create(\_key) | 创建并返回指定key的table，如果已存在返回null | table |
| ModData.create() | 使用随机UUID创建table,返回随机key | String |
| ModData.getOrCreate(\_key) | 创建或返回指定key的table | table |
| ModData.get(\_key) | 返回指定key的table | table |
| ModData.exists(\_key) | 返回指定Key的table是否已创建 | boolean |
| ModData.remove(\_key) | 删除指定key的table，返回table | table |
| ModData.add(\_key,\_table) | 覆盖指定key的table |  |

默认GlobalModData不会在客户端与服务端同步数据，需要手动调用指定函数。

|  |  |  |
| --- | --- | --- |
| 函数 | 说明 | 返回 |
| ModData.transmit(\_key) | 在服务端调用，指定key的table会发送到所有客户端。在客户端调用，指定key的table会发送到服务端。 |  |
| ModData.request(\_key) | 仅限客户端调用，指定key的table会发送到服务端。 |  |

当客户端和服务端收到table时并不会直接覆盖对应key的数据，只会触发OnReceiveGlobalModData事件，你需要根据自己的需求来实现覆盖、仅解析、只覆盖部分等操作。

## 客户端与服务端互相调用

### sendClientCommand和sendServerCommand

要想实现客户端和服务端的互相调用，需要使用sendClientCommand和sendServerCommand全局函数。当收到命令时会触发OnServerCommand和OnClientCommand事件，在事件回调中根据模块名和命令名判断是否是需要解析的命令并执行相应的操作。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sendServerCommand | 在服务端发送命令 | String——模块名 | String——命令名 | table——命令数据 |  |
| sendServerCommand | 在服务端发送命令 | IsoPlayer——玩家 | String——模块名 | String——命令名 | table——命令数据 |
| sendClientCommand | 在客户端发送命令 | String——模块名 | String——命令名 | table——命令数据 |  |
| sendClientCommand | 在客户端发送命令 | IsoPlayer——玩家 | String——模块名 | String——命令名 | table——命令数据 |

客户端在每次按下按键时发送命令，服务端接收到后输出对应玩家姓名和按键码。服务端每两分钟发送一次命令，所有客户端收到命令并打印"2 minutes passed"。注意：在单机时sendClientCommand和sendServerCommand不会生效，如果你需要mod同时支持单机和联机，请参考下面写法。

### LuaNet

LuaNet是游戏对sendClientCommand和sendServerCommand的封装，简化一些操作。用LuaNet重写上面的实现。

## 定时操作 ISBaseTimedAction

定时操作指需要读条的操作，比如抽烟、合成物品、健身等等操作。

### 新建定时操作

这是最基础的新建模板，如果你新建的操作依赖其它的数据，可以在new中传入。示例中的stopOnWalk等属性是在LuaTimedActionNew中需要使用的，你可以选择模板中那样覆盖，也可以不声明使用默认值。示例中的值就是默认值。

### 实现定时操作

新建的定时操作只有基础功能，即显示进度条。下一步就是实现实现定时操作具体效果。只需要重写ISBaseTimedAction的函数即可。

### 示例：抽烟

## 设置

### 沙盒设置

通过media/sandbox-options.txt可以方便快捷的自定义MOD的沙盒设置选项。基础格式参照下方。

Sandbox\_CN.txt

|  |  |  |  |
| --- | --- | --- | --- |
| 通用属性 | 说明 | 属性值类型 | 取值范围 |
| type | 选项类型。 | string | integer、boolean、double、enum、string |
| page | 选项所在菜单栏。会显示对应语言的Sandbox\_【page】的值。 | string |  |
| translation | 选项显示名称。会显示对应语言的Sandbox\_【translation】的值。 | string |  |
| boolean类型属性 |  |  |  |
| default | 默认值。 | boolean |  |
| double类型属性 |  |  |  |
| min | 最小值。 | double |  |
| max | 最大值。 | double |  |
| default | 默认值。 | double |  |
| integer类型属性 |  |  |  |
| min | 最小值。 | integer |  |
| max | 最大值。 | integer |  |
| default | 默认值。 | integer |  |
| string类型属性 |  |  |  |
| default | 默认值。 | string |  |
| enum类型属性 |  |  |  |
| numValues | 枚举值数量。 | integer |  |
| default | 默认值。 | integer |  |
| valueTranslation | 枚举值显示名称。会显示对应语言的Sandbox\_【valueTranslation】translation\_option【数字】的值。 | string |  |

### 游戏设置

目前，游戏设置的设计目的是只影响客户端的设置且不与单局游戏相关联，参考游戏原有设置项。如果你需要设置与单局游戏关联，你应该考虑使用沙盒设置，如果你依然打算使用用游戏设置实现，你需要自己通过4.12实现设置的保存加载机制。如果你需要设置在服务端和客户端同步，或者客户端使用服务端设置，你需要使用4.9自己实现同步逻辑。

如果你不想自己实现设置的保存和加载、服务端与客户端的同步，你可以使用别人提供好的前置mod，比如[Mod Options (Build 41)](https://steamcommunity.com/sharedfiles/filedetails/?id=216943599)。使用方法可以在mod简介里寻找，这里就不展开赘述。

#### 分割线

#### 下拉选择框

#### 勾选框

#### 自定义

除了游戏提供快捷添加函数，还可以使用mainOptions.mainPanel:addChild(\_child)添加自定义UI控件，只要使用 local gameOption = GameOption:new('child', \_child)相关联就行。

#### 获得设置值

## 文件保存和读取

在游戏中可以使用moddata保存数据，但如果你需要在未进行游戏时保存数据，比如保存上一节提到的菜单设置值。你就要保存到硬盘本地文件。但需要你自己决定保存的格式和解析对应内容。

### 保存文件

### 读取文件

在实际使用中我们经常一次性读取文件所有内容。

## 刷新

当你完成物品、车辆的制作。你会发现在游戏中并不会刷新，因为你还没为其添加刷新定义。

### 物品刷新

[容器列表](https://docs.qq.com/sheet/DYkxtdnhJWXlGWEJq)

### 车辆刷新

[停车位列表](https://docs.qq.com/sheet/DYlJEUk5SZFhPcUVR)

# 扩展

## 遍历List

部分函数的返回是Java中的List，不能直接使用for pairs和for ipairs遍历。

## 5.2. Git工具