HTML5 魔塔样板使用指南

制作：艾之葵

目录

一、 概述

众所周知，魔塔的趋势是向移动端发展，贴吧中也常常能见到“求手机魔塔”的帖子。然而现有的工具中，NekoRPG 有着比较大的局限性，游戏感较差，更是完全没法在 iOS 上运行。而一些 APP 的魔塔虽然可用，但是必须要下载安装，对于 Android 和 iOS 还必须开发不同的版本，非常麻烦。

但是，现在我们有了 HTML5。HTML5 的画布（canvas）以及它被 Android/iOS 内置浏览器所支持的特性，可以让我们做出真正意义上的全平台覆盖的魔塔。

事实上，在贴吧的试水发布也证明了，H5 魔塔确实是可以成功的。两部即使是复刻的魔塔也受到了不少人的追捧，其流畅的手感和全平台支持的特性，也让很多没办法打开电脑的人爱不释手。

然而，一般而言使用非 RMXP 制作魔塔往往需要一定的编程技术，HTML5 魔塔自然也不例外。但是，为了能让大家更加注重于“做塔”本身，而不用考虑做塔以外的各种脚本问题，我特意制作了这样一部 HTML5 的魔塔样板。

这个魔塔样板，可以让你在完全不懂任何编程的情况下，做出自己的 H5 魔塔。不会代码？没关系！只要你想做，就能做出来！

下面，我将详细地介绍，如何使用这一个样板来制作属于自己的 HTML5 魔塔。

二、 快速上手：现在就做一部塔！

在这一节中，将详细介绍做一部塔的流程。现在，让我们来做一部单层塔！

2.1 前置需求

你需要有满足如下条件才能进行制作：

- Windows 8 以上操作系统；Windows 7 需要安装.Net Framework 4.0。
  （能打开同目录下的“地图生成器.exe”即可）
- 任一款现代浏览器。强烈推荐 Chrome。
- 一个很好的文本编辑器。推荐带有高亮染色、错误提示等效果。例如：WebStorm，VSCode，或者至少也要 Sublime Text。
  （Sublime Text 下载地址：https://www.sublimetext.com/ ，如提示注册的话百度随便搜索一个注册码输入即可。）
- RPG Maker XP，任一个魔塔样板（推荐魔塔样板 7630）

只要满足了上述条件，你就可以开始做自己的塔啦！

2.2 新建一个剧本

类似于 RMXP，本塔每层楼都是一个“剧本”，剧本内主要定义了本层的地图和各种事件。主函数将读取每个剧本，并生成实际的地图供游戏使用。

我们打开 libs/floors/ 目录，这个目录是所有剧本的目录。我们需要指定一个楼层名，例如 MT1；然后，我们可以将 MT0.js（模板）复制重命名为为 MT1.js，并使用文本编辑器打开。（参见下页的图）。

然后将楼层名改为 MT1，floorId 改名为 MT1；title 可以改成任意内容，将在切换楼层时进行显示（比如可以改成“1 层小塔”）。

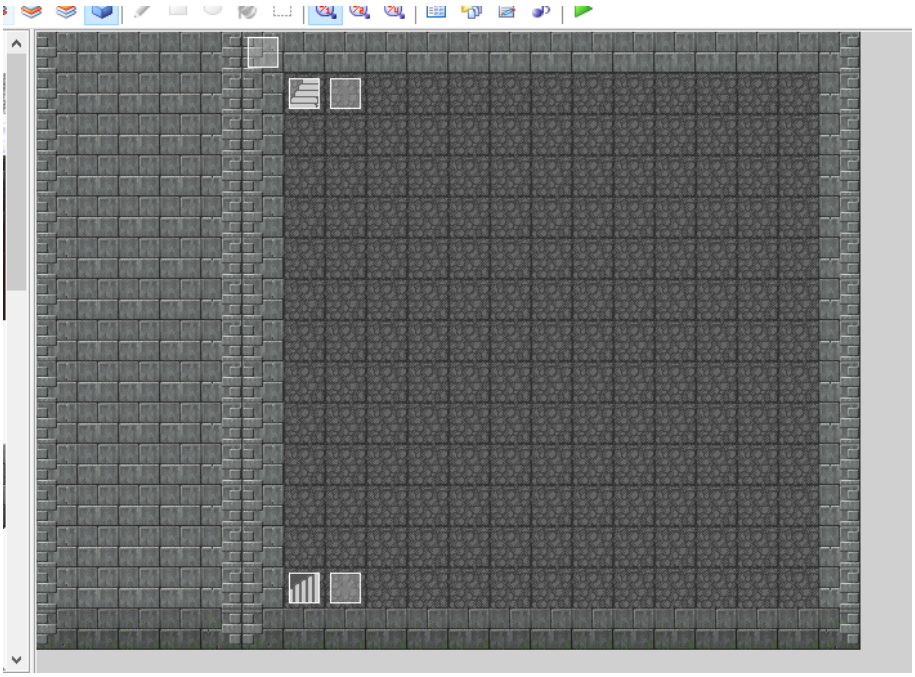
具体样板文件的每个要素都有详细的注释。我们最终的任务其实是，将每个楼层的剧本（地图&事件）给写完即可。

换句话说，只需要简单的复制操作，我们就可以新建一个剧本了。

2.3 绘制地图

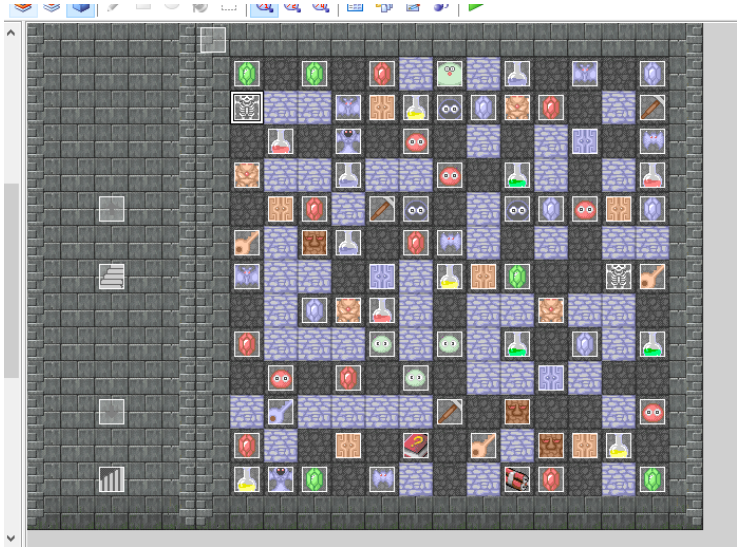
遗憾的是，我们的样板是没有像 RMXP 那样有着很好的 UI 界面，供大家直接进行绘图可视化操作的。然而，我们仍然可以利用已有的 RMXP 和魔塔样板，绘制好地图，然后利用目录中的“地图生成器”来转成样板所识别的格式。

首先，我们打开 RMXP 和魔塔样板，来到绘制地图页面。



然后，任意绘制一张地图。

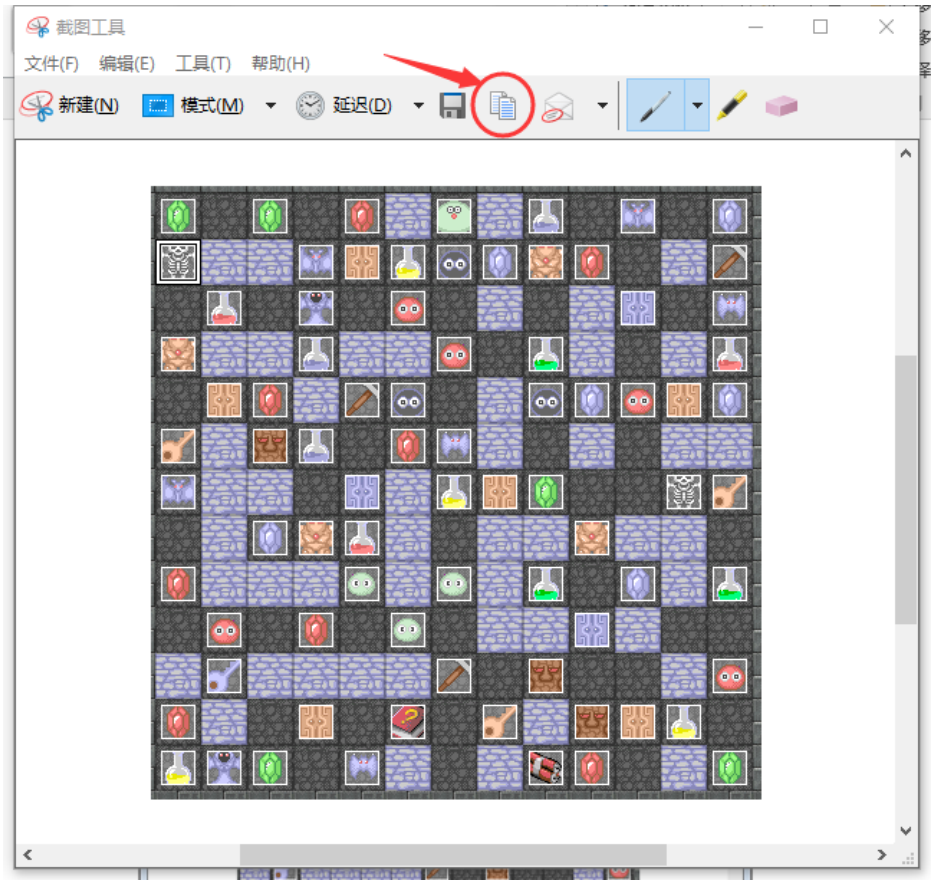
在这里我们就以 1 层小塔的地图为例。你也可以任意绘制自己的地图。



（我把原塔素材改成了经典样式，但是本质上是一样的。）

请注意，我们无需编辑任何事件。只需要让地图“看起来长成这个样子”就行。

当绘制好需要的地图后，我们打开 Windows 自带的“截图工具”，并将整个地图有效区域截图下来，并将其复制到剪切板。



截图时请注意：只截取有效游戏空间内数据，并且有效空间内的范围必须是 13*13。（如果地图小于 13*13，请用星空或墙壁填充到 13*13）。

确认地图的图片文件已经复制到剪切板后，我们打开“地图生成器”，并点“加载图片”。大约 1-2 秒后，可以得到地图的数据。

如果有识别不一致的存在，即生成的地图和实际的地图不符，则需要在左边的输入框内实际手动修改，然后再点“图片生成”即可。有关每个数字对应的图块名称，请参见 images 目录下的 meaning.txt

注：地图生成器默认只支持经典素材。如果有自定义素材需求（例如原版的 1 层小塔那种素材），请参见 5.1。

经过确认，生成的地图和原始地图保持一致后，点击“复制地图”，然后粘贴到刚刚剧本文件里的 maps 中。

```
    "canFlyTo": true, // 该楼能否被楼传器飞到（不能的话在该楼也不允许使用楼传器）
    "map": [ // 地图数据，需要是13x13，建议使用地图生成器来生成
        [29,   0,    29,   0,    27,   3,    204,  3,    32,   0,    206,  0,    28],
        [209,  3,    3,    206,  81,   34,   203,  28,   221,  27,   0,    3,    47],
        [0,    31,   0,    217,  0,    202,  0,    3,    0,    3,    82,   0,    205],
        [221,  3,    3,    32,   3,    3,    202,  0,    33,   3,    0,    3,    31],
        [0,    81,   27,   3,    47,   203,  0,    3,    203,  28,   202,  81,   28],
        [21,   3,    215,  32,   0,    27,   205,  3,    0,    3,    0,    3,    3],
        [206,  3,    3,    0,    82,   3,    34,   81,   29,   0,    0,    209,  21],
        [0,    3,    28,   221,  31,   3,    0,    3,    3,    221,  3,    3,    0],
        [27,   3,    3,    3,    201,  3,    201,  3,    33,   0,    28,   3,    33],
        [0,    202,  0,    27,   0,    201,  0,    3,    3,    82,   3,    0,    0],
        [3,    22,   3,    3,    3,    47,   0,    215,  0,    0,    3,    202],
        [27,   3,    0,    81,   0,    45,   0,    21,   3,    215,  81,   34,   0],
        [34,   217,  29,   0,    205,  3,    0,    3,    49,   27,   0,    3,    29],
    ],
    "firstArrive": [ // 第一次到该楼层触发的事件
```

通过这种在 RMXP 中画图，截图复制，再用地图生成器识别的方式，我们成功将我们需要的地图变成了样板可识别的格式。

2.4 录入数据，开始游戏

有了地图后，我们下一步需要做的就是录入数据。数据主要包括如下几种：

- 勇士初始的属性
- 全局变量（宝石效果、全局 Flag 等）
- 怪物数据（每个怪物的攻防血金币经验等等）

下面依次进行说明。

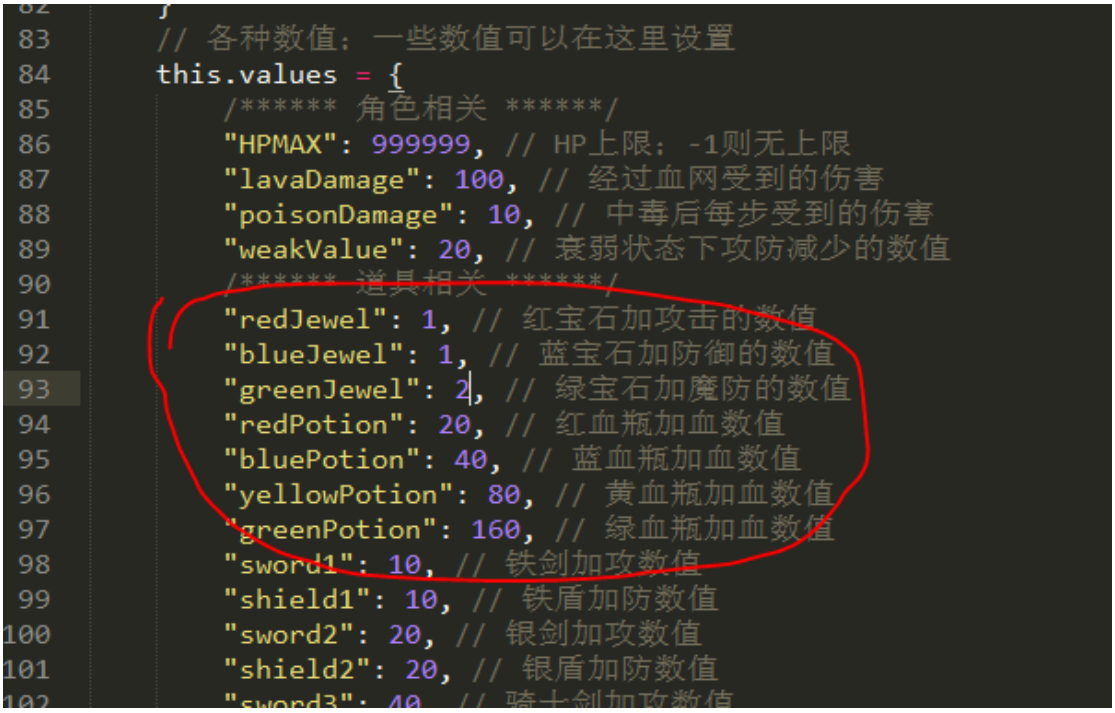
我们打开 data.js 文件，这里面定义了各种全局属性和勇士初始值。

我们可以将本塔标题改名为“1 层小塔”，游戏的唯一标识符叫 onefloor，然后可以直接修改勇士的各项初始数据。

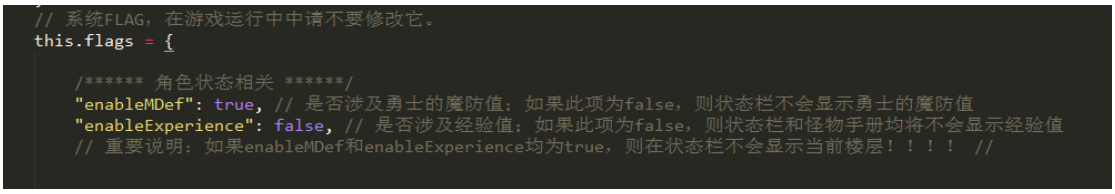
```
data.prototype.init = function() {
    this.firstData = {
        "title": "1层小塔", // 游戏名，将显示在标题页面以及切换楼层的界面中
        "name": "onefloor", // 游戏的唯一英文标识符。由英文、数字、下划线组成，不能超过20个字符。
        "version": "Ver 1.0.0 (Beta)", // 当前游戏版本；版本不一致的存档不能通用。
        "floorId": "MT1", // 初始楼层ID
        "hero": { // 勇士初始数据
            "id": "hero1", // 此项关系到icons中的heroID，一般不要改
            "name": "勇士", // 勇士名；可以改成喜欢的
            "hp": 500, // 初始生命值
            "atk": 10, // 初始攻击
            "def": 10, // 初始防御
            "mdef": 0, // 初始魔防
            "money": 0, // 初始金币
            "experience": 0, // 初始经验
            "items": { // 初始道具个数
                "keys": {
                    "yellowKey": 0,
                    "blueKey": 0,
                    "redKey": 0
                },
                "constants": {},
                "tools": {}
            },
            "flyRange": [], // 初始可飞的楼层；一般留空数组即可
            "loc": {"direction": "up", "x": 6, "y": 12}, // 勇士初始位置
            "flags": { // 游戏过程中的变量或flags
                "poison": false, // 毒
                "weak": false, // 衰
                "curse": false, // 咒
```

请注意，勇士的初始位置一栏，x 为横坐标，y 为纵坐标；即，x 为从左到右第几列，y 为从上到下第几行，均从 0 开始计算。

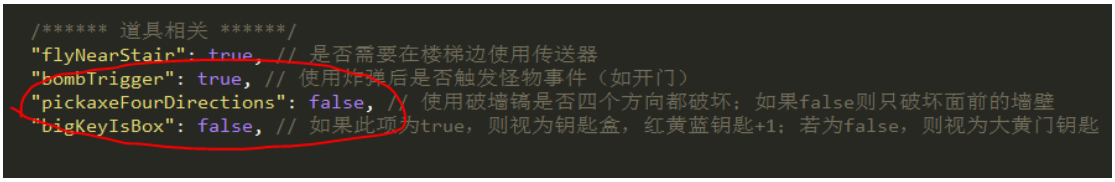
修改完初始化信息后，接下来我们需要修改道具的信息（比如宝石加攻防的数值，血瓶加生命的数值等）。还是在这个 data.js 文件，往下拉，找到 values 一项，并进行相应的设置。



然后，再设置一些系统 Flag，以进行游戏。继续将 data.js 往下拉，我们注意到本塔是存在魔防的，不存在经验，因此我们可以简单地将 enableMDef 改为 true，enableExperience 改成 false。

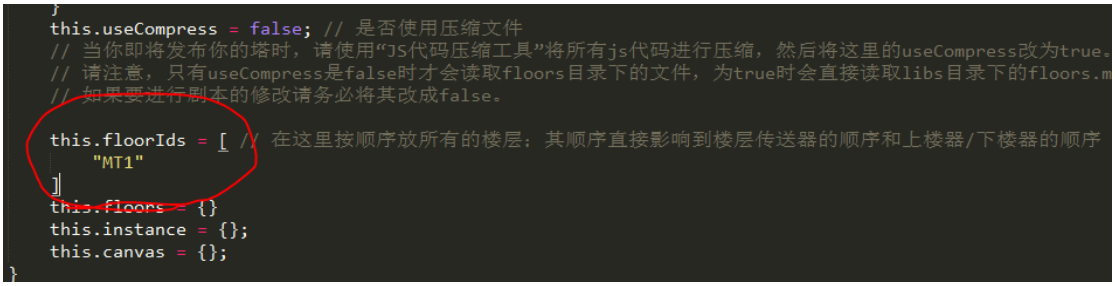


同理，本塔的破墙镐只能破面前的墙壁，因此 pickaxeFourDirections 需要改成 false。



其他的几项暂时不会被涉及到，因此不用考虑。

全局变量修改完毕后，我们需要告诉主函数加载该楼层。打开 main.js（该文件和 index.html 同级），找到 this.floorIds 项，将其值改为楼层 ID 即 MT1。

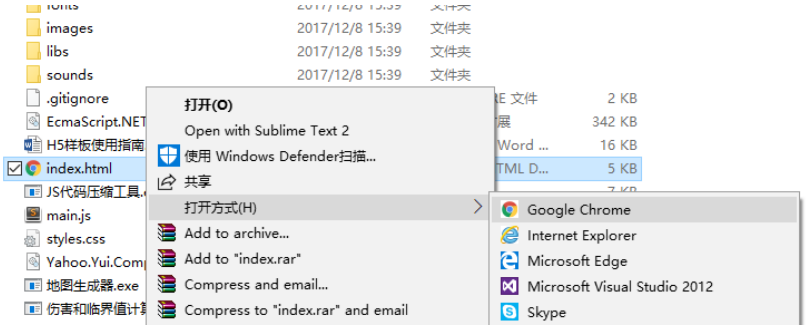


最后一步就是录入怪物数据。打开 enemys.js 文件，依次输入你在本塔内使用到的所有怪物的攻防血的数据。其中怪物的特殊属性（special 项）与该文件下面的 getSpecialText 对应。

```
enemys.prototype.init = function () {
    // 怪物属性初始化定义
    this.enemys = {
        'greenSlime': {'name': '绿头怪', 'hp': 60, 'atk': 25, 'def': 2, 'money': 0, 'experience': 0, 'special': 0},
        'redSlime': {'name': '红头怪', 'hp': 38, 'atk': 70, 'def': 2, 'money': 0, 'experience': 0, 'special': 1}, // 先攻
        'blackSlime': {'name': '青头怪', 'hp': 100, 'atk': 70, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'slimelord': {'name': '怪王', 'hp': 130, 'atk': 120, 'def': 5, 'money': 0, 'experience': 0, 'special': 0},
        'bat': {'name': '小蝙蝠', 'hp': 80, 'atk': 40, 'def': 2, 'money': 0, 'experience': 0, 'special': 4}, // 2连击
        'bigBat': {'name': '大蝙蝠', 'hp': 125, 'atk': 40, 'def': 8, 'money': 0, 'experience': 0, 'special': 0},
        'redBat': {'name': '红蝙蝠', 'hp': 0, 'atk': 120, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'vampire': {'name': '冥灵魔王', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'skeleton': {'name': '骷髅人', 'hp': 110, 'atk': 40, 'def': 4, 'money': 0, 'experience': 0, 'special': 0},
        'skeletonSoilder': {'name': '骷髅士兵', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'skeletonCaptain': {'name': '骷髅队长', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'ghostSkeleton': {'name': '冥队长', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'zombie': {'name': '兽人', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'zombieKnight': {'name': '兽人武士', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'rock': {'name': '石头人', 'hp': 70, 'atk': 14, 'def': 11, 'money': 0, 'experience': 0, 'special': 3},
        'slimeMan': {'name': '影子战士', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0}, // 模仿怪的攻防
        'bluePriest': {'name': '初级法师', 'hp': 90, 'atk': 30, 'def': 4, 'money': 3, 'experience': 0, 'special': 2},
        'redPriest': {'name': '高级法师', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'brownWizard': {'name': '初级巫师', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0, 'value': 0},
        'redWizard': {'name': '高级巫师', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0, 'value': 0},
        'yellowGuard': {'name': '初级卫兵', 'hp': 90, 'atk': 15, 'def': 12, 'money': 0, 'experience': 0, 'special': 3},
        'blueGuard': {'name': '中级卫兵', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'redGuard': {'name': '高级卫兵', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'swordsman': {'name': '双手剑士', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'soldier': {'name': '冥战士', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'yellowKnight': {'name': '金骑士', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
        'redKnight': {'name': '红骑士', 'hp': 0, 'atk': 0, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
```

只需要修改自己用到的怪物属性即可，其他没有用到的怪物完全无所谓。

做完后保存所有文件，然后右键，选择使用 chrome 浏览器打开 index.html，就能立刻看到自己的塔并开始游戏啦！是不是很简单呢！



2.5 压缩与发布

当你将上述步骤完成后，你实际上已经做出来了一个魔塔，并且可以发布给大家进行游戏了。

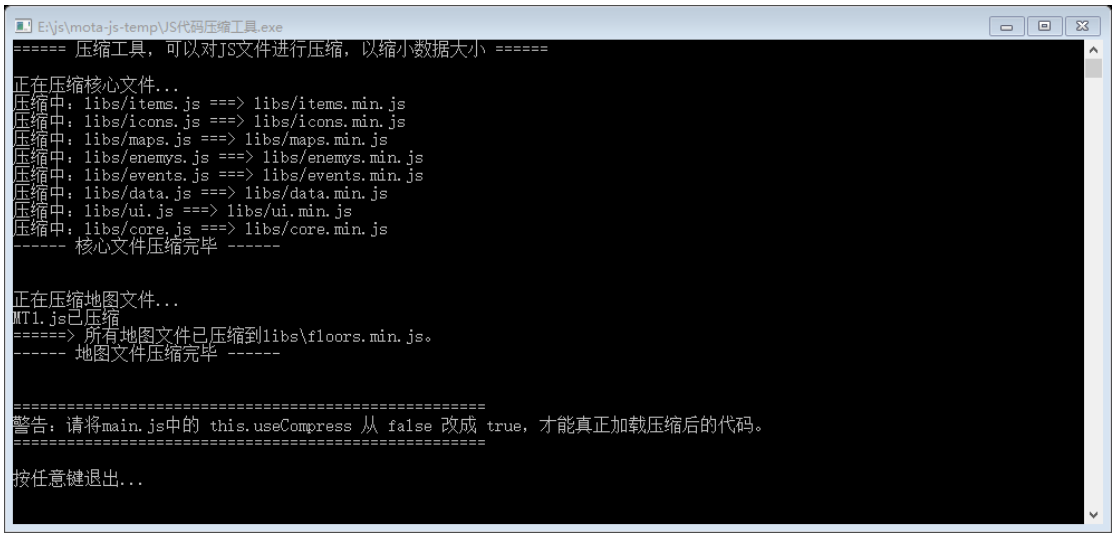
目前而言发布有如下几种方式：

- 离线版本：直接将游戏文件夹打包，放到百度网盘等地方供用户下载；用户下载到本地后直接使用浏览器打开 index.html 进行游戏。
  - 手机端的部分浏览器如 chrome 也支持本地网页，可以下载到手机然后直接打开进行游戏。
- 在线版本：将游戏放到某个服务器上，大家在线联网游戏。

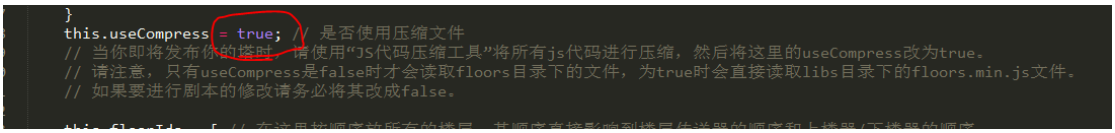
离线版本的好处是：先全部下载后再游戏，无需考虑流量的问题，也可以支持高清音乐的播放。坏处是：没办法在多平台之间迁移（平台同步的锅），而浏览器打开本地文件有丢失存档的风险。

在线版本的好处是：随时随地可以玩，可以多平台接档，还可以在后台看到一些统计信息，了解大概有多少人进行了游戏（如果需要）；坏处是需要一个服务器，且还要考虑到用户流量的问题。

在此我们只讨论在线版本。当你决定发布游戏时，强烈建议先将 JS 代码进行压缩以节省可能的 IO 请求以及网络流量。直接打开同目录下的“JS 代码压缩工具”进行压缩即可。



压缩完毕后，你还需要将 main.js 中的 useCompress 从 false 改为 true，这样方才只会加载压缩后的文件。



如果你需要发布到服务器上且你没有服务器，可以直接将本塔发给我（艾之葵），我会给负责你部署上去的。我的服务器至少能保证两年内有效。

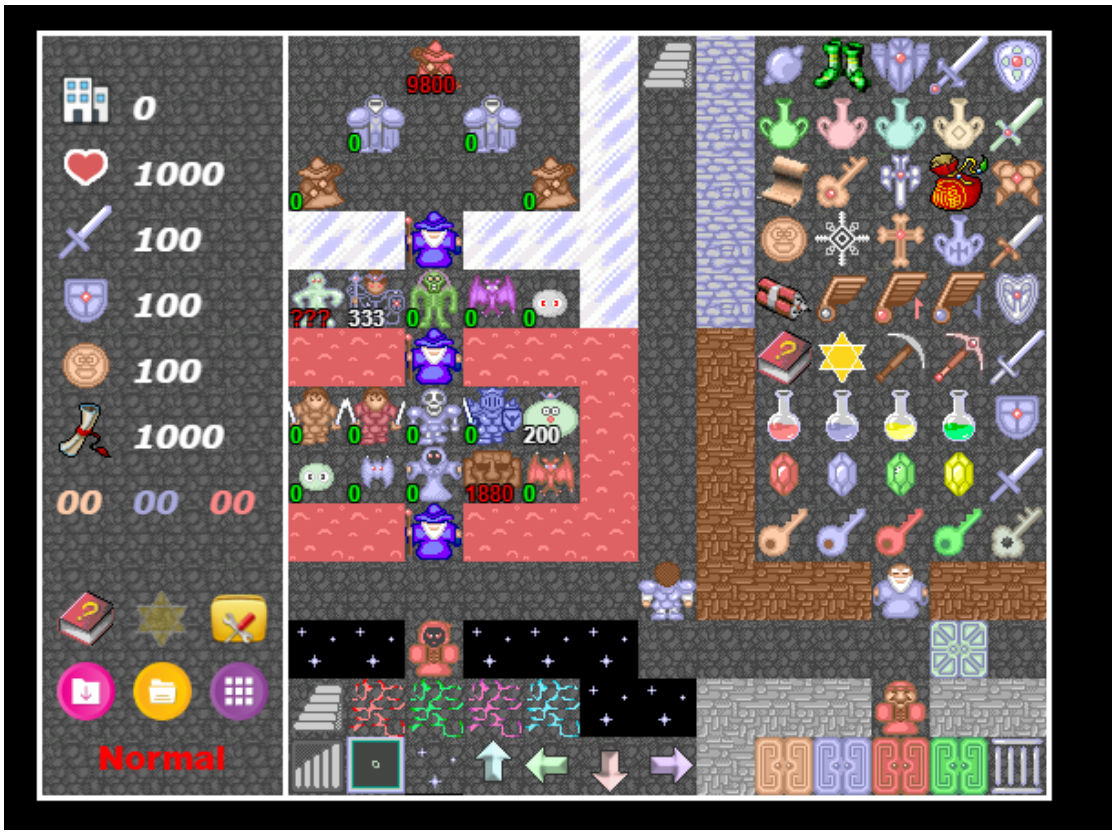
然后就是发布帖子、链接二维码，能让任何人在任何时候任何平台上都能进行游戏啦！是不是很简单呢！

在下面的几章里，将对样板的各个元件、事件等依次进行介绍。

三、 元件说明

在本章中，将对样板里的各个元件进行说明。各个元件主要包括道具、门、怪物、楼梯等等。

请打开样板 0 层（sample0.js）进行参照对比。



3.1 道具

本塔目前支持的所有道具列表在样板 0 层中已全部给出。当你在样板 0 层中拿到某个宝物时会有提示，这里不再赘述，详见拿到该道具的说明。

大多数宝物都有默认的效果，十字架和屠龙匕首暂未定义，如有自己的需求可参见 5.2 的内容。

请注意，本塔没有“装备”的说法，所有剑盾拿到后将立刻作为攻防数值直接加到勇士的属性上。

拿到道具后将触发 afterGetItem 事件，有关事件的详细介绍请参见第四章。

如需修改某个道具的效果，在不同区域宝石数据发生变化等问题，请参见 5.2 的说明。

3.2 门

本塔支持 6 种门，黄蓝红绿铁花。前五种门需要有对应的钥匙打开，花门只能通过调用 openDoor 事件进行打开。

本塔支持暗墙，但是暗墙也必须通过 openDoor 事件开启。例如，样板 2 层的小偷事件，就是可以打开一个暗墙的。

开门后可触发该层的 afterOpenDoor 事件，有关事件的详细介绍请参见第四章。

3.3 怪物

本塔支持的怪物列表参见 enemys.js。其与 images 目录下的 enemys.png 素材按顺序一一对应。如不知道怪物素材长啥样的请打开 enemys.png 对比查看。

如有自己的怪物素材需求请参见 5.1 的内容。

怪物可以由特殊属性，每个怪物最多只能有一个特殊属性。怪物的特殊属性所对应的数字（special）在下面的 getSpecialText 中定义，请勿对已有的属性进行修改。

```
77
78▼ enemys.prototype.getSpecialText = function (enemyId) {
79      if (enemyId == undefined) return "";
80      var special = this.enemys[enemyId].special;
81      if (special == 1) return "先攻";
82      if (special == 2) return "魔攻";
83      if (special == 3) return "坚固";
84      if (special == 4) return "2连击";
85      if (special == 5) return "3连击";
86      if (special == 6) return "4连击";
87      if (special == 7) return "破甲";
88      if (special == 8) return "反击";
89      if (special == 9) return "净化";
90      if (special == 10) return "模仿";
91      if (special == 11) return "吸血";
92      if (special == 12) return "中毒";
93      if (special == 13) return "衰弱";
94      if (special == 14) return "诅咒";
95      if (special == 15) return "领域";
96      if (special == 16) return "夹击";
97      return "";
98  }
99
```

怪物的伤害计算在下面的 calDamage 函数中，如有自己需求的伤害计算公式请修改该函数的代码。

如果 data.js 中的 enableExperience 为 false，即不启用经验的话，怪物手册里将不显示怪物的经验值，打败怪物也不获得任何经验。

拿到幸运金币后，打怪获得的金币将翻倍。

吸血怪需要在怪物后添加 value，代表吸血的比例。

```
, 'def': 0, 'money': 0, 'experience': 0, 'special': 0},
': 120, 'def': 0, 'money': 17, 'experience': 0, 'special': 16},
, 'atk': 120, 'def': 0, 'money': 12, 'experience': 0, 'special': 11, 'value': 1/3}, // 吸血怪需要在后面添加value代表吸血比例
': 120, 'def': 0, 'money': 15, 'experience': 0, 'special': 14},
```

中毒怪让勇士中毒后，每步扣减的生命值由 data.js 中的 values 定义。

衰弱怪让勇士衰弱后，攻防会暂时下降一定的数值（直到衰弱状态解除恢复）；这个下降的数值同在 data.js 中的 values 定义。

```
    /****** 角色相关 ******/
    "HPMAX": 999999, // HP上限，-1则无上限
    "lavaDamage": 100, // 经过血网受到的伤害
    "poisonDamage": 10, // 中毒后每步受到的伤害
    "weakValue": 20, // 衰弱状态下攻防减少的数值
    /****** 道具相关 ******/
    "redJewel": 1, // 红宝石加攻击的数值
```

诅咒怪将让勇士陷入诅咒状态，诅咒状态下杀怪不获得金币和经验值。

领域怪需要在怪物后添加 value，代表领域伤害的数值。如果勇士生命值扣减到 0，则直接死亡触发 lose 事件。

```
: 0, 'money': 3, 'experience': 0, 'special': 2},
'money': 0, 'experience': 0, 'special': 0},
': 0, 'money': 16, 'experience': 0, 'special': 15, 'value': 100}, // 领域怪需要加value表示领域伤害的数值
': 0, 'money': 160, 'experience': 0, 'special': 15, 'value': 200},
': 0, 'money': 10, 'experience': 0, 'special': 0},
'money': 0, 'experience': 0, 'special': 0},
```

出于游戏性能的考虑，我们不可能每走一步都对领域和夹击进行检查。因此我们需要在本楼层的 checkBlock 中指明哪些点可能会触发领域和夹击事件，在这些点才会对领域和夹击进行检查和处理。

```
  },
  "checkBlock": [
    /****** 领域、夹击检查事件 ******/
    // 所有可能的领域、夹击点均需要在这里给出，否则将不会触发检查事件
    // 另外，如果该点已经存在events事件或changeFloor事件（即上面有相同点位置定义），则会被覆盖
    // afterBattle, afterGetItem, afterOpenDoor则不受影响（仍能正常工作）。
    // 所以 |****** 强烈要求可能的夹击、领域点不要存在自定义事件！！ ******|
    "1,0", "3,0", "0,1", "2,1", "4,1", "1,2", "3,2"
  ]
```

请注意这里的”x,y”代表该点的横坐标为 x，纵坐标为 y；即从左到右第 x 列，从上到下的第 y 行（从 0 开始计算）。

本塔不支持阻击、激光、仇恨、自爆、退化等属性。

（这些属性基本都太恶心了，恶心到都不想加上去）

如有额外需求，可参见 5.3，里面讲了如何设置一个新的怪物属性。

3.4 路障、楼梯、传送门

血网的伤害数值、中毒后每步伤害数值、衰弱时暂时攻防下降的数值，都在 data.js 的 values 内定义。

路障同样会尽量被自动寻路绕过。

有关楼梯和传送门，必须在该层样板的 changeFloor 里指定传送点的目标。

请注意这里的”x,y”代表该点的横坐标为 x，纵坐标为 y；即从左到右第 x 列，从上到下的第 y 行（从 0 开始计算）。如(6,0)代表最上面一行的正中间一列。

floorId 指定的是目标楼层的唯一标识符（ID）。

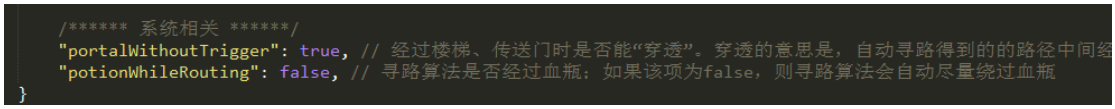
后面可以写 stair 到 upFloor 或 downFloor，表示将前往目标楼层的上楼梯/下楼梯位置。你也可以写 loc 然后指定目标点的坐标。

请注意的是，如果目标楼层有多个楼梯，写 stair 可能会导致到达的楼梯不确定，这时候请使用 loc 方式来指定具体的点位置。

可以指定 direction 为 up/left/right/down，指定后勇士将面向该方向。

可以指定 time，指定后切换动画时长为指定的数值。

楼梯和传送门默认可“穿透”。所谓穿透，就是当寻路穿过一个楼梯/传送门后，不会触发楼层传送事件，而是继续前进。通过系统 Flag 可以指定是否穿透，你也可以对每个传送点单独设置该项。



上面就是整个样板中的各个元件说明。通过这种方式，你就已经可以做出一部没有任何事件的塔了。

尝试着做一个两到三层的塔吧！

四、 事件

本章内将对样板所支持的事件进行介绍。

4.1 事件的机制

本塔所有的事件都是依靠触发（trigger）完成的。例如，勇士碰到一个门可以触发一个事件（openDoor），勇士碰到怪物可以触发一个事件（battle），勇士碰到一个（上面定义的）楼层传送点可以触发一个事件（changeFloor），勇士穿过路障可以触发一个事件（passNet），包括勇士到达一个指定的 checkBlock 也可以触发一个检查领域、夹击的事件。上面说的这些事件都是系统本身自带的，即类似于 RMXP 中的公共事件。

```
events.prototype.init = function () {
    this.events = {
        'battle': function (data, core, callback) {
            core.battle(data.event.id, data.x, data.y);
            if (core.isset(callback))
                callback();
        },
        'getItem': function (data, core, callback) {
            core.getItem(data.event.id, 1, data.x, data.y);
            if (core.isset(callback))
                callback();
        },
        'openDoor': function (data, core, callback) {
            core.openDoor(data.event.id, data.x, data.y, true);
            if (core.isset(callback))
                callback();
        },
        'changeFloor': function (data, core, callback) {
            var heroLoc = null;
            if (core.isset(data.event.data.loc)) {
                heroLoc = {'x': data.event.data.loc[0], 'y': data.event.data.loc[1]};
                if (core.isset(data.event.data.direction))
                    heroLoc.direction = data.event.data.direction;
            }
            core.changeFloor(data.event.data.floorId, data.event.data.stair,
                heroLoc, data.event.data.time, callback);
        },
        'passNet': function (data, core, callback) {
            core.events.passNet(data);
            if (core.isset(callback))
                callback();
        },
```

上述这些默认的事件已经存在处理机制，不需要我们操心。我们真正所需要关心的，其实只是一个自定义的事件。

本塔中的所有自定义事件能且只能被其他事件触发。不存在 RMXP 里面那种，设置了某个变量为 true 后，一个事件被自动执行的问题。这点和 RMXP 的区别非常大，请务必注意。

所有事件都存在两种状态：启用和禁用。启用状态下，该事件才处于可见状态，可被触发、交互与处理。禁用状态下该事件相当于不存在，不可见、不可被触发、不可交互。所有事件默认情况下都是启用的，除非指定了 enable: false。在事件列表中使用 type: show 和 type: hide 可以将一个禁用事件启用，或将一个启用事件给禁用。这些都在后面会提到。

4.2 自定义事件

打开样板 1 层（sample1.js）有着一些介绍。下面是更为详细的说明。

所有自定义的事件都是如下的写法：

```
"events": { // 该楼的所有可能事件列表

    "x,y": {
        "trigger": "action", // 触发的trigger: action代表是个自定义事件
        "enable": true, // 该事件初始状态下是否处于启用状态
        "data": [ // 实际执行的事件列表
            // 事件1
            // 事件2
            // ...
        ]
    }

},
```

这里的”x,y”代表该点的横坐标为 x，纵坐标为 y；即从左到右第 x 列，从上到下的第 y 行（从 0 开始计算）。

我们上面提到，有很多系统已经默认的事件（例如开门、打怪等，相当于公共事件）。如果我们需要自定义一个事件，则需要”trigger”: “action”，它表示该点是一个自定义事件。

如果系统本身存在事件（如一个怪物），且你指定了”trigger”: “action”，则原事件会被覆盖。这种情况下一般需采用后面的 afterBattle，afterOpenDoor 和 afterGetItem 来进行事件的处理。

如果该点本身不存在系统事件，则”trigger”:”action”可被省略不写：

```
"x,y": {
    // 除非你要覆盖该点已经存在的的系统默认事件，否则"trigger": "action"可以省略
    "enable": true, // 该事件初始状态下是否处于启用状态
    "data": [ // 实际执行的事件列表
        // 事件1
        // 事件2
        // ...
    ]
}
```

“enable”: true 代表该点初始状态下是否是启用的。如果 enable 为 false，则该点初始状态下禁用，将不会被显示和交互（比如如果该点是个怪，指定了 enable 为 false，则该怪不会显示在地图上，也不会发生战斗）。

默认情况下 enable 是 true，所以如果 enable 为 true，该项也可以省略不写：

```
"x,y": {
    // 除非你要覆盖该点已经存在的的系统默认事件，否则"trigger": "action"可以省略
    // 如果该事件初始是启用状态，则可以省略"enable": true；如果禁用状态必须加上"enable": false
    "data": [ // 实际执行的事件列表
        // 事件1
        // 事件2
        // ...
    ]
}
```

“data”为实际执行的事件列表。类似于 RMXP 中的“脚本”，也是由一系列事件顺序构成的（其中可以使用 if 和 choices 来进行条件判断或用户选择，后面会具体提到）。

如果大括号里只有”data”，则可以省略大括号和”data”，直接写中括号数组，换句话说，上面和下面这种写法也是等价的，可以进行一下比较：

```
// 如果大括号里只有"data"项（没有"action"或"enable"），则可以省略到只剩下中括号
"x,y": [ // 实际执行的事件列表
    // 事件1
    // 事件2
    // ...
]
```

这种简写方式可以极大方便地造塔者进行造塔。

请注意：如果该点初始的 enable 为 false，或者该点本身有系统默认事件且需要覆盖（trigger），则必须采用上面那种大括号写的方式来定义。

“data”中，是由一系列的自定义事件类型组成。每个元素类似于：

```
// 如果大括号里只有"data"项（没有"action"或"enable"），则可以省略到只剩下中括号
"x,y": [ // 实际执行的事件列表
    {"type": "xxx", ...}, // 事件1
    {"type": "xxx", ...}, // 事件2
    ...
    // 按顺序写事件，直到结束
]
```

“type”为该自定义事件的类型；而后面的...则为具体的一些事件参数。

每次，系统都将取出数组中的下一个事件，并进行处理；直到数组中再无任何事件，才会完全结束本次自定义事件，恢复游戏状态。

下面将依次对所有自定义事件类型进行介绍。请根据黑体红字进行索引。

- **text：显示一段文字（剧情）**

使用{“type”:”text”}可以显示一段文字。后面”data”可以指定文字内容。

```
// 如果大括号里只有"data"项（没有"action"或"enable"），则可以省略到只剩下中括号
"x,y": [ // 实际执行的事件列表
    {"type": "text", "data": "在界面上一段文字"}, // 显示文字事件
    {"type": "text", "data": "这是第二段文字"}, // 显示第二个文字事件
    ...
    // 按顺序执行事件，直到结束
]
```

该项可以简写成直接的字符串的形式，即下面这种方式也是可以的：

```
// 如果大括号里只有"data"项（没有"action"或"enable"），则可以省略到只剩下中括号
"x,y": [ // 实际执行的事件列表
    "在界面上一段文字", // 直接简写，和{"type":"text", "data": ...}完全等价
    {"type": "text", "data": "这是第二段文字"}, // 显示第二个文字事件
    ...
    // 按顺序执行事件，直到结束
]
```

所有文字事件均可以进行简写，系统会自动转成{“type”: “text”}的形式。

另外非常值得注意的一点就是，我们需要手动输入\n 来进行换行：

```
// 如果大括号里只有"data"项（没有"action"或"enable"），则可以省略到只剩下中括号
"x,y": [ // 实际执行的事件列表
    "在界面上一段文字", // 直接简写，和{"type":"text", "data": ...}完全等价
    "这是第一行\n这是第二行\n这是第三行", // 显示第二个文字事件
    ...
    // 按顺序执行事件，直到结束
]
```

我们可以给文字加上标题或图标，只要以\t[…]开头就可以，大致共有如下几种情况：

- \t[hero] 显示勇士的图标和名字
- \t[monster_id] 显示某个怪物的图标和名字。monster_id 在 enemys 中有定义，请前往参照。
  - 例如：\t[blackMagician] 将显示黑暗大法师的图标和名字。
- \t[名字,npc_id] 显示某个 NPC 的名字和图标。npc_id 所对应的图标具体在 icons.js 中有定义，请前往参照。
  - 例如：\t[小妖精,fairy] 将显示名字为“小妖精”，且是仙子的图标。
- \t[标题] 直接显示标题。
  - 如果该中括号内只有一项，且不为 hero 也不为某个怪物的 ID，则会直接显示。如 \t[你死了] 直接显示一个标题为“你死了”。

```
"x,y": [ // 实际执行的事件列表
    "一段普通文字",
    "\t[hero]这是一段勇士说的话",
    "\t[blackMagician]这是一段黑暗大法师说的话",
    "\t[小妖精,fairy]这是一段小妖精说的话，使用仙子的图标",
    "\t[你赢了]直接显示标题为“你赢了”"
]
```

另外值得一提的是，我们是可以在文字中计算一个表达式的值的。只需要将表达式用 ${ 和 } 整个括起来就可以。

```
"x,y": [ // 实际执行的事件列表
    "1+2=${1+2}, 4*5+6=${4*5+6}", // 显示"1+2=3, 4*5+6=26"
]
```

我们可以使用 status:xxx 代表勇士的一个属性值；item:xxx 代表某个道具的个数；flag:xxx 代表某个自定义的变量或 flag 值。

```
"x,y": [ // 实际执行的事件列表
    "你的当前攻击力是${status:atk}，防御力是${status:def}",
    "你的攻防和的十倍是${10*(status:atk+status:def)}",
    "你的红黄蓝钥匙总数为${item:yellowKey+item:blueKey+item:redKey}",
    "你访问某个老人的次数为${flag:man_times}", // 取用一个自定义变量/flag
    // ...
]
```

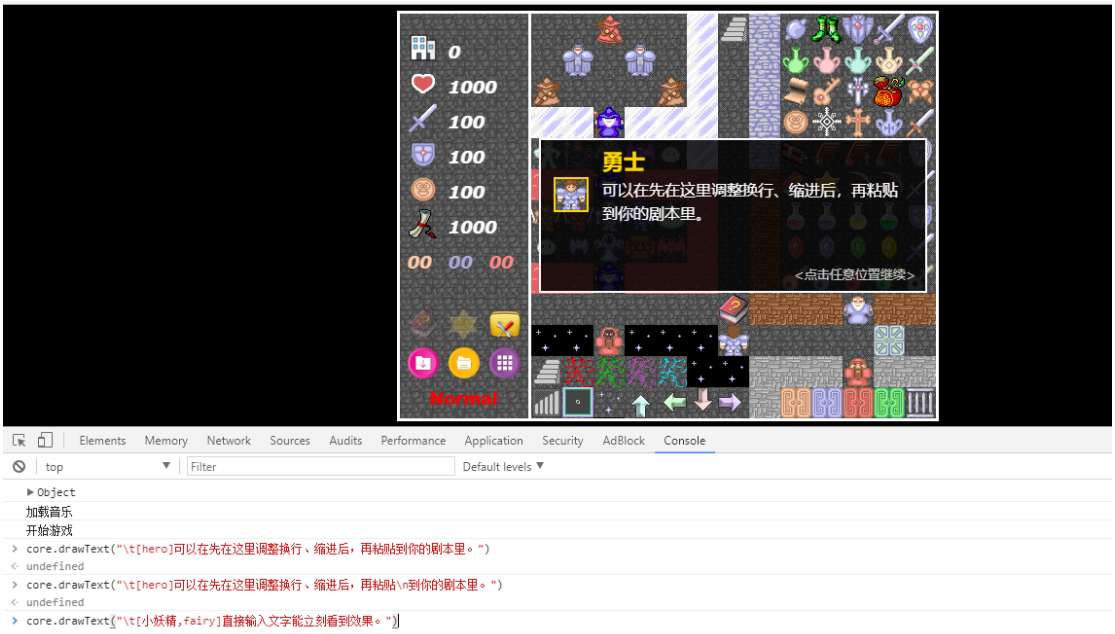
status: xxx 取勇士属性时只能使用如下几个：hp（生命值），atk（攻击力），def（防御力），mdef（魔防值），money（金币），experience（经验）。

item: xxx 中的 xxx 为道具 ID。所有道具的 ID 定义在 items.js 中，请自行查看。例如，item:centerFly 代表中心对称飞行器的个数。

flag: xxx 中的 xxx 为一个自定义的变量/Flag；如果没有对其进行赋值则默认值为 false。

另外，有个小 trick。是否有时候会觉得不好找到\n（换行）的位置？有一个很简单的方法：

你可以用 Chrome 浏览器打开游戏，按 Ctrl+Shift+I 打开开发者工具，找到 Console（控制台），并中输入 core.drawText(“…”) 即可立刻看到文字显示的效果。适当添加\n，使得显示效果满意后，再复制粘贴到你的剧情文本中。



- **setValue：设置勇士的某个属性、道具个数，或某个变量/Flag 的值**

{“type”: “setValue”} 能修改勇士的某个属性、道具个数、或某个自定义变量或 Flag 的值。

其大致写法如下：

```
"x,y": [ // 实际执行的事件列表
    {"type": "setValue", "name": "status:atk", "value": "status:atk+10"}, // 攻击提高10点
    {"type": "setValue", "name": "status:money", "value": "1000"}, //将金币数值设为1000（不是+1000）
    {"type": "setValue", "name": "status:hp", "value": "status:hp*2"}, // 生命翻倍
    {"type": "setValue", "name": "item:yellowKey", "value": "item:yellowKey+3"}, // 黄钥匙个数+3
    {"type": "setValue", "name": "item:bomb", "value": "item:bomb+10"}, // 炸弹个数+10
    {"type": "setValue", "name": "flag:man_times", "value": "0"}, // 将变量man_times设成0
    {"type": "setValue", "name": "flag:man_times", "value": "flag:man_times+2*status:atk"},
    // 将变量man_times的值加上勇士攻击数值的两倍
    ...
]
```

使用 setValue 需要指定 name 和 value 选项。

name 为你要修改的属性/道具/Flag，每次只能修改一个值。写法和上面完全相同，status:xxx 表示勇士一个属性，item: xxx 表示某个道具个数，flag:xxx 表示某个变量或 flag 值。参见上面的介绍。

value 是一个表达式，将通过这个表达式计算出的结果赋值给 name。该表达式同样可以使用 status:xxx, item:xxx, flag:xxx 的写法表示勇士当前属性，道具个数和某个变量/Flag 值。

- **show: 将一个禁用事件启用**

我们上面提到了，所有事件都必须靠其他事件驱动来完成，不存在当某个 flag 为 true 时自动执行的说法。那么，我们自然要有启用事件的写法。

使用{“type”:”show”}可以将一个本身禁用的事件启用。

```
"x,y": [ // 实际执行的事件列表
    {"type": "show", "loc": [3,6], "floorId": "MT1", "time": 500}, // 启用MT1层[3,6]位置事件，动画500ms
    {"type": "show", "loc": [3,6], "time": 500}, // 如果启用的目标是当前层，则可以忽略floorId选项
    {"type": "show", "loc": [3,6]}, // 如果不指定time选项，则会立刻显示；否则动画效果逐渐显示，time为时间。
    ...
]
```

show 事件需要用 loc 指定目标点的坐标；剩下有两个参数 floorId 和 time。

floorId 为目标点的楼层，如果不是该楼层的事件（比如 4 楼小偷开 2 楼的门）则是必须的，如果是当前楼层可以忽略不写。

time 为动画效果时间，如果指定了某个大于 0 的数，则会以动画效果慢慢从无到有显示，动画时间为该数值；如果不指定该选项则无动画直接立刻显示。

要注意的是，调用 show 事件后只是让该事件从禁用状态变成启用，从不可见不可交互变成可见可交互，但本身不会去执行该点的事件。

- **hide: 将一个启用事件禁用**

{“type”:”hide”}和 show 刚好相反，它会让一个已经启用的事件被禁用。

其参数和 show 也完全相同，loc 指定事件的位置，floorId 为楼层（同层可忽略），time 指定的话事件会以动画效果从有到无慢慢消失。

但是和 show 事件有所区别的是：loc 选项也可以忽略；如果忽略 loc 则使当前事件禁用。（即使禁用当前事件，也不会立刻结束当前正在进行的，而是仍然会依次将列表中剩下的事件执行完）

请注意，一次性事件必须要加 {“type”:”hide”}，尤其是例如走到某个点，触发对话或机关门（陷阱）这种，否则每次都会重复触发。

NPC 对话事件结束后如果需要 NPC 消失也需要调用 {“type”: “hide”}，可以不写 loc 选项代表当前事件，可以指定 time 使 NPC 动画消失。

```
"x,y": [ // 实际执行的事件列表
    {"type": "hide", "loc": [3,6], "floorId": "MT1", "time": 500}, // 禁用MT1层[3,6]位置事件，动画500ms
    {"type": "hide", "loc": [3,6], "time": 500}, // 如果禁用的目标是当前层，则可以忽略floorId选项
    {"type": "hide", "loc": [3,6]}, // 如果不指定time选项，则会立刻消失；否则动画效果逐渐消失，time为时间。
    {"type": "hide", "time": 500}, // 不指定loc选项则默认为当前点；例如这个就是500ms消失当前对话的NPC
    {"type": "hide"}, // 无动画将当前事件禁用，常常适用于某个空地点（触发陷阱事件启用机关门这种）
]
```

- **trigger: 立即触发另一个地点的事件**

{“type”:”trigger”} 会立刻触发当层另一个地点的自定义事件。

上面我们说到，show 事件会让一个禁用事件启用且可被交互；但是如果我想立刻让它执行应该怎么办呢？使用 trigger 就行。

其基本写法如下：

```
"x,y": [ // 实际执行的事件列表
    {"type": "trigger", "loc": [3,6]}, // 立即触发"3,6"点的事件；当前剩下的事件全部不再执行
    "执行trigger后，这段文字不会再被显示"
]
```

其后面带有 loc 选项，代表另一个地点的坐标。

执行 trigger 事件后，当前事件将立刻被结束，剩下所有内容被忽略；然后重新启动另一个地点的 action 事件。例如上面这个例子，下面的文字将不会再被显示，而是直接跳转到”3,6”对应的事件列表从头执行。

- **revisit: 立即重启当前事件**

revisit 和 trigger 完全相同，只不过是立刻触发的还是本地点的事件

```
"x,y": [ // 实际执行的事件列表
    {"type": "revisit"}, // 立即重新触发本事件；等价于 {"type": "trigger", "loc": [x,y]}
    "执行revisit后，这段文字不会再被显示"
]
```

revisit 其实是 trigger 的简写，只不过是 loc 固定为当前点。

revisit 常常使用在一些商人之类的地方，当用户购买物品后不是离开，而是立刻重新访问重新进入购买页面。

- **exit: 立刻结束当前事件**

上面说到像商人一类，购买物品后可以立刻 revisit 重新访问，但是这样就相当于陷入了死循环导致无法离开。

可以使用{“type”:”exit”}立刻结束事件。调用 exit 后，将立刻结束一切事件，清空事件列表，并返回游戏。

例如玩家点击商人的“离开”选项，则可以调用 exit 返回游戏。

```
"x,y": [ // 实际执行的事件列表
    {"type": "exit"}, // 立即结束事件并恢复游戏，一切列表中的事件都不会再被执行
    "执行exit后，这段文字不会再被显示"
]
```

- **update: 立刻更新状态栏和地图显伤**

当我们在上面调用 show 事件，显示一个怪物后，该怪物将不会有显伤显示。

如果你需要刷新状态栏和地图显伤，只需要简单地调用 {“type”: “update”} 即可。

- **sleep: 等待多少毫秒**

  等价于 RMXP 中的“等待 x 帧”，不过是以毫秒来计算。

  基本写法：{“type”: “sleep”, “time”: xxx} ，其中 xxx 为指定的毫秒数。

```
"x,y": [ // 实际执行的事件列表
    {"type": "sleep", "time": 1000}, // 等待1000毫秒
    "等待1000毫秒后才会执行这个事件"
]
```

- **battle: 强制战斗**

  调用 battle 可强制与某怪物进行战斗（而无需去触碰到它）

  例如，《宿命的旋律》中，一区有个骷髅队长，当你拿了它周围三个物品时，就会立刻触发强制战斗事件。这时候就可以用{“type”: “battle”} 实现。

  其基本写法是： {“type”: “battle”, “id”: xxx}，其中 xxx 为怪物 ID。

```
// 注意：初级王宫打死后的开门事件是在 afterBattle 中调用
"10,4": [ // 开门后走进去的事件：强制战斗
    "\t[blackKing]你终于还是来了。",
    "\t[hero]放开我们的公主！",
    "\t[blackKing]如果我不愿意呢？",
    "\t[hero]无需多说，拔剑吧！",
    {"type": "battle", "id": "blackKing"}, // 强制战斗
    // 如果战斗失败直接死亡，不会继续触发接下来的剧情。
    {"type": "hide", "loc": [10,2]}, // 战斗后需要手动使怪物消失；战斗后不会引发afterBattle事件。
    {"type": "openDoor", "loc": [8,7]}, // 开门口的机关门
    "\t[blackKing]没想到你已经变得这么强大了... 算你厉害。\n公主就交给你了，请好好对她。",
    {"type": "hide"} // 隐藏本事件
],
```

  上面就是样板层中右上角的强制战斗例子。

  如果强制战斗失败，则会立刻生命归 0 并死亡，调用 lose 函数，接下来的事件不会再被执行。

  强制战斗没有指定 loc 的选项，因此战斗后需要调用 hide 使怪物消失（如果有必要）。强制战斗不会触发任何 afterBattle 里的事件。

- **openDoor: 开门**

  调用{“type”:”openDoor”}可以打开一扇门。

```
"x,y": [ // 实际执行的事件列表
    {"type": "openDoor", "loc": [3,6], "floorId": "MT1"}, // 打开MT1层的[3,6]位置的门
    {"type": "openDoor", "loc": [4,5]}, // 如果是本层可忽略floorId
]
```

loc 指定门的坐标，floorId 指定门所在的楼层 ID。如果是当前层则可以忽略 floorId 选项。

如果 loc 所在的点是一个墙壁，则作为暗墙来开启。

如果 loc 所在的点既不是门也不是墙壁，则忽略忽略本事件。

openDoor 不会触发任何 afterOpenDoor 里的事件。

- **changeFloor：楼层切换**

在事件中也可以对楼层进行切换。一个比较典型的例子就是 TSW 中，勇士在三楼的陷阱被扔到了二楼，就是一个楼层切换事件。

changeFloor 的事件写法大致如下。

```
"x,y": [ // 实际执行的事件列表
    {"type": "changeFloor", "floorId": "sample0", "loc": [10,10], "direction": "left", "time": 1000},
    // 切换楼层floorId为目标楼层名，loc为目标坐标（不支持stair），direction可选为勇士方向，time可选为切换时间
]
```

可以看到，与上面的楼梯、传送门的写法十分类似。

但是相比那个而言，不支持 stair 楼梯位置（只能写坐标），没有穿透选项。

direction 为可选的，指定的话将使勇士的朝向变成该方向

time 为可选的，指定的话将作为楼层切换动画的时间。

- **changePos：当前位置切换**

有时候我们不想要楼层切换的动画效果，而是直接让勇士从 A 点到 B 点。

这时候可以用 changePos。其参数和 changeFloor 类似，但少了 floorId 和 time 两个选项。

```
"x,y": [ // 实际执行的事件列表
    {"type": "changePos", "loc": [10,10], "direction": "left"},
    // 直接切换勇士坐标。loc为目标地点，direction可选为勇士切换后朝向
]
```

- **openShop：打开一个全局商店**

使用 openShop 可以打开一个全局商店。有关全局商店的说明可参见 4.3。

```
"1,0": [ // 金币商店
    // 打开商店前，你也可以添加自己的剧情
    // 例如，通过if来事件来判断是不是第一次访问商店，是的则显示一段文字（类似宿命的华音那样）
    {"type": "openShop", "id": "moneyShop1"} // 这里的id要和data.js中你定义的商店ID完全一致
    // 调用openShop事件后，所有当前事件都会被结束（同exit事件），然后打开一个全局商店
],
"5,0": [ // 经验商店
    {"type": "openShop", "id": "expShop1"}
],
```

- **move: 让某个 NPC/怪物移动**

如果我们需要移动某个 NPC 或怪物，可以使用{“type”: “move”}。

下面是该事件常见的写法：

```
"x,y": [ // 实际执行的事件列表
    {"type": "move", "time": 750, "steps": [ // 动画移动效果，time为每步事件（毫秒），steps是个数组指定了移动的方位
        {"direction": "right", "value": 2}, // 向右移动两步，再向下移动一步，并消失
        "down" // 如果该方向只移动一步，可以直接这样简写。 这种写法等价于： {"direction":"down","value":1}
    ], "immediateHide": true}, // immediateHide可选，如果指定并为true则立刻消失，否则
]
```

time 选项必须指定，为每移动一步所需要用到的时间。

steps 为一个数组，其每一项为一个 {“direction” : xxx, “value”: n}，表示该步是向 xxx 方向移动 n 步。

如果只移动一步可以直接简单的写方向字符串（up/left/down/right）。

immediateHide 为一个可选项，代表该事件移动完毕后是否立刻消失。如果该项指定了并为 true，则移动完毕后直接消失，否则以动画效果消失。

值得注意的是，当调用 move 事件时，实际上是使事件脱离了原始地点。为了避免冲突，规定：move 事件会自动调用 hide 事件。

换句话说，当 move 事件被调用后，该点本身的事件将被禁用。

move 完毕后移动的 NPC/怪物一定会消失，只不过可以通过 immediateHide 决定是否立刻消失还是以 time 作为时间来动画效果消失。

如果想让 move 后的 NPC/怪物仍然可以被交互，需采用如下的写法：

```
"4,3": [ // [4,3]是一个NPC，比如小偷
    {"type": "move", "time": 750, "steps": [ // 向上移动两格，每步750毫秒，移动完毕立刻消失
        {"direction": "up", "value": 2},
    ], "immediateHide": true},
    {"type": "show", "loc": [4,1]} // 指定[4,1]点的NPC立刻生效（显示）
    {"type": "trigger", "loc": [4,1]} // 立刻触发[4,1]点的事件
],
"4,1": { // [4,1]也是这个NPC，而且是向上移动两个的位置
    "enable": false, // 初始时需要是禁用状态，被show调用后将显示出来
    "data": [
        "\t[杰克,thief]这样看起来就好像移动过去后也可以被交互"
    ]
}
```

即，在移动的到达点指定一个初始禁用的相同 NPC，然后 move 事件中指定 immediateHide 使立刻消失，并 show 该到达点坐标使其立刻显示（看起来就像没有消失），然后就可以触发目标点的事件了。

- **playSound: 播放音效**

使用 playSound 可以立刻播放一个音效。

使用方法：{“type”: “playSound”, “name”: “item.ogg”}

值得注意的是，如果是额外添加进文件的音效，则需在 main.js 中 this.sounds 里加载它。

由于考虑到用户流量问题，每个额外音效最好不超过 20KB。

- **win: 获得胜利**

{“type”: “win”, “reason”: “xxx”} 将会直接调用 events.js 中的 win 函数，并将 reason 作为参数传入。

该事件会显示获胜页面，并重新游戏。

- **lose: 游戏失败/Game Over**

{“type”: “lose”, “reason”: “xxx”} 将会直接调用 events.js 中的 lose 函数，并将 reason 作为参数传入。

该事件会显示失败页面，并重新开始游戏。

- **if: 条件判断**

使用{“type”: “if”}可以对条件进行判断，根据判断结果将会选择不同的分支执行。

其大致写法如下：

```
"x,y": [
    {"type": "if", "condition": "...", // 测试某个条件
        "true": [ // 成立的场合
            // 如果条件成立，将接着执行这个列表中的事件
        ],
        "false": [ // 不成立的场合
            // 如果条件不成立，将接着执行这个列表中的事件
        ]
    }
]
```

我们可以在 condition 中给出一个表达式（能将 status:xxx, item:xxx, flag:xxx 来作为参数），并进行判断是否成立。

如果条件成立，则将继续执行”true”中的列表事件内容。

如果条件不成立，则将继续执行”false”中的列表事件内容。

例如下面这个例子，每次将检查你的攻击力是否大于 500，不是的场合将给你的攻击力加 100 点。

```
"x,y": [ // 实际执行的事件列表
    {"type": "if", "condition": "status:atk>500", // 条件判断：攻击力是否大于500
        "true": [  // 如果该条件为真，将继续执行这个列表中的事件
            "你的攻击力已经大于500了！",
            {"type": "exit"} // 结束
        ],
        "false": [ // 如果该条件为假，将继续执行这个列表中的事件
            "你的当前攻击力是${status:atk}，不足500！\n给你增加100点攻击力！",
            {"type": "setValue", "name": "status:atk", "value": "status:atk+100"} // 攻击+100
            // 接着会接着if语句后执行，即revisit事件
        ]
    },
    {"type": "revisit"} // 立刻重启本事件，直到攻击大于500后exit结束
]
```

需要额外注意的几点：

- 给定的表达式（condition）一般需要返回 true 或 false。
- flag:xxx 可取用一个自定义变量或 flag。如果从未设置过该 flag，则其值默认为 false。而 JS 中，false==0 这个判断是成立的，因此我们可以简单使用 “flag:npc_times==0” 来判断某个 NPC 是否被访问过。
- 即使成功失败的场合不执行事件，对应的 true 或 false 数组也需要存在，不过简单的留空就好。
- if 可不断嵌套，一层套一层；如成立的场合再进行另一个 if 判断等。
- if 语句内的内容执行完毕后将接着其后面的语句继续执行。

- **choices: 给用户提供选项**

choices 是一个很麻烦的事件，它将弹出一个列表供用户进行选择。

当用户做出了不同的选择，可以有着不同的分支处理。

其完全类似于 RMXP 中的“显示选择项”，“XX 的场合”，只不过同样是需要使用数组来定义。

其大致写法如下：

```
"x,y": [
    {"type": "choices", "text": "...", // 提示文字
        "choices": [
            {"text": "选项1文字", "action": [
                // 选项1执行的事件

            ]},
            {"text": "选项2文字", "action": [
                // 选项2执行的事件

            ]},
            {"text": "选项3文字", "action": [
                // 选项3执行的事件

            ]},
        ]
    },
]
```

其中最外面的”text”为提示文本。同上面的”type”:”text”一样，支持${}表达式的计算，和\t 显示名称、图标。text 可省略，如果省略将不显示任何提示文字。

choices 为一个数组，其中每一项都是一个选项列表。

每一项的 text 为显示在屏幕上的选项名，也支持${}的表达式计算，但不支持\t[]的显示。action 为当用户选择了该选项时将执行的事件。

选项可以有任意多个，但一般不要超过 6 个，否则屏幕可能塞不下。

下面是一个卖钥匙的事件，是一个比较复杂却也较为典型的 if 和 choices 合并使用的样例。

```
"x,y": [
    {"type": "choices", "text": "\t[商人,woman]少年，你需要钥匙吗？\n我有大把的！", // 商人提示文字
        "choices": [
            {"text": "黄钥匙（20金币）", "action": [ // 选项1：黄钥匙
                {"type": "if", "condition": "status:money>=20",  // 一个if事件，判断金币够不够
                    "true": [ // 足够了：扣减金币，增加黄钥匙
                        {"type": "setValue", "name": "status:money", "value": "status:money-20"},
                        {"type": "setValue", "name": "item:yellowKey", "value": "item:yellowKey+1"}
                    ],
                    "false": [ // 不够，显示提示文字
                        "\t[商人,woman]你的金币不足！"
                    ]
                },
                // 会继续执行下面的revisit事件
            ]},
            {"text": "蓝钥匙（50金币）", "action": [ // 选项2：蓝钥匙
                // 和上面的黄钥匙选项相似，这里就不重复写了
            ]},
            // 还可以添加红钥匙的选项
            {"text": "离开", "action": [ // 选项3：离开
                {"type": "exit"} // 离开商店：不会继续执行下面的的revisit事件。
            ]},
        ]
    },
    {"type": "revisit"} // 立刻重启本事件：也就是购买钥匙后不会离开商店
]
```

- **function: 自定义 JS 脚本**

上述给出了这么多事件，但有时候往往不能满足需求，这时候就需要执行自定义脚本了。

```
"x,y": [
    {"type": "function", "function": function() { // 执行一段JS脚本
        // 在这里写JS脚本代码
        alert(core.getStatus("atk")); // 弹窗显示勇士的攻击力
    }},
]
```

{“type”:”function”}需要有一个”function”参数，它是一个 JS 函数，里面可以写任何自定义的 JS 脚本；系统将会执行它。

系统所有支持的 API 都在附录中给出。

这里只简单列出给一些最常见的 API：

- core.getStatus(name) 获得勇士的某个属性（hp/atk/def/…）
- core.setStatus(name, value) 设置勇士某个属性值为 value
- core.itemCount(name) 获得某个道具的个数
- core.getItem(name, count) 获得某个道具 count 个
- core.setItem(name, value) 设置某个道具为 value 个
- core.getFlag(name, defaultValue) 获得某个自定义变量/flag；如果未定义则返回 defaultValue
- core.setFlag(name, value) 将某个自定义变量/flag 设置为 value
- core.hasFlag(name) 判断某个自定义 flag 是否成立。只有被被赋值过，且不为 0 或 false 时才会返回 true。
- core.updateStatusBar() 立刻更新状态栏和地图显伤。（上面各种 get 和 set 均不会对状态栏和地图显伤更新，需要手动调用这个函数。）
- core.insertAction(list) 往当前事件列表中插入一系列事件。使用这个函数插入的事件将在这段自定义 JS 脚本执行完毕后立刻执行。
- ……

4.3 全局商店

我们可以采用上面的 choices 方式来给出一个商店。这样的商店确实可以有效地进行操作，但是却是“非全局”的，换句话说，只有在碰到 NPC 的时候才能触发商店事件。

我们可以定义“全局商店”，其可以直接被快捷栏中的“快捷商店”进行调用。换句话说，我们可以定义快捷商店，让用户在任意楼层都能快速使用商店。

全局商店定义在 data.js 中，找到 shops 一项。

```
],
"shops": { // 定义全局商店（即快捷商店）
    "moneyShop1": { // 商店唯一ID
        "name": "贪婪之神", // 商店名称（标题）
        "icon": "blueShop", // 商店图标，blueShop为蓝色商店，pinkShop为粉色商店
        "textInList": "3楼金币商店", // 在快捷商店栏中显示的名称
        "use": "money", // 商店所要使用的。只能是"money"或"experience"。
        "need": "20+10*times*(times+1)",  // 商店需要的金币/经验数值；可以是一个表达式，以times作为参数计算。
        // 这里用到的times为该商店的已经的访问次数。首次访问该商店时times的值为0。
        // 上面的例子是50层商店的计算公式。你也可以写任意其他的计算公式，只要以times作为参数即可。
        // 例如：  "need": "25" 就是恒定需要25金币的商店；  "need": "20+2*times" 就是第一次访问要20金币，以后每次递增2金币的
        // 如果是对于每个选项有不同的计算公式，写 "need": "-1" 即可。可参见下面的经验商店。
        "choices": [ // 商店的选项
            {"text": "生命+800", "effect": "status:hp+=800"},
            // 如果有多个effect以分号分开，参见下面的经验商店
            {"text": "攻击+4", "effect": "status:atk+=4"},
            {"text": "防御+4", "effect": "status:def+=4"},
            {"text": "魔防+10", "effect": "status:mdef+=10"}
            // effect只能对status和items进行操作，不能修改flag值。
            // 中间只能用+=符号（也就是只能增加某个属性或道具）
            // 其他effect样例：
            // "items:yellowKey+=1" 黄钥匙+1
            // "items:pickaxe+=3" 破墙镐+3
        ]
    },
    "expShop1": { // 商店唯一ID
```

全局商店全部定义在 data.js 中的 shops 一项里。

每个全局商店有一个唯一标识符（ID），然后是一系列对该商店的定义。

- name 为商店的名称（打开商店后的标题）
- icon 为商店的图标，blueShop 为蓝色商店，pinkShop 为粉色商店
- textInList 为其在快捷商店栏中显示的名称，如“3 楼金币商店”等
- use 为消耗的类型，是金币（money）还是经验（experience）。
- need 是一个表达式，计算商店所需要用到的数值。
  - 可以将 times 作为参数，times 为该商店已经访问过的次数，第一次访问时 times 是 0。
  - 如果对于每个选项都需要不同的数值，这里设为”-1”；可参见下面经验商店的例子。
- choices 为商店的各个选项，是一个 list，每一项是一个选项
  - text 为显示文字。请注意这里不支持 ${} 的表达式计算。
  - effect 为该选项的效果；effect 只能对 status 或 items 进行操作，且必

须是 status:xxx+=yyy 或 item:xxx+=yyy 的形式。即中间必须是+=符号。

- 如有多个 effect（例如升级全属性提升），使用分号分开，参见经验商店的写法。

像这样定义了全局商店后，即可在快捷栏中看到。

请注意，快捷商店默认是不可被使用的。直到至少调用一次自定义事件中的{“type”: “openShop”} 打开商店后，才能真正在快捷栏中被使用。

4.4 系统引发的自定义事件

我们知道，所有自定义事件都是需要定义在”x,y”处，并且得让用户经过或撞上才能触发的。

但是有一系列的事件，例如战斗、获取道具、开门等，是系统已经预先设定好的事件，我们不能将其覆盖为自定义事件，否则原本的战斗等事件会被覆盖。

为了解决此问题，在每层的剧本中引入了三个元素：afterBattle, afterGetItem, afterOpenDoor。

当某个战斗结束后，将执行 afterBattle 中，对应位置的事件。

当获取某个道具后，将执行 afterGetItem 中，对应位置的事件。

当开了某个门后，将执行 afterOpenDoor 中，对应位置的事件。

例如，下面就是一个典型的杀怪开门的例子。每当杀死一个守卫机关门的怪物，将检查是否满足打开机关门的条件。如果是，则开启机关门。

```
},
"afterBattle": { // 战斗后可能触发的事件列表
    "9,6": [ // 初级卫兵1
        {"type": "setValue", "name": "flag:door", "value": "flag:door+1"}, // 将"door"这个自定义flag加一
        {"type": "if", "condition": "flag:door==2", // 一个条件判断事件，条件是"door"这个flag值等于2
            "true": [ // 如果条件成立：打开机关门
                {"type": "openDoor", "loc": [10,5]}
            ],
            "false": [] // 如果条件不成立则无事件触发
        },
    ],
    "11,6": [ // 初级卫兵2：注意由于打怪顺序问题，可能都得写一遍。
        {"type": "setValue", "name": "flag:door", "value": "flag:door+1"}, // 将"door"这个自定义flag加一
        {"type": "if", "condition": "flag:door==2", // 一个条件判断事件，条件是"door"这个flag值等于2
            "true": [ // 如果条件成立：打开机关门
                {"type": "openDoor", "loc": [10,5]}
            ],
            "false": [] // 如果条件不成立则无事件触发
        },
    ],
},
"afterGetItem": { // 获得道具后可能触发的事件列表
```

同样，为了实现类似于 RMXP 中，到达某一层后自动触发某段事件的效果，样板中还存在 firstArrive 事件。当且仅当勇士第一次到达某层时，将会触发此事件。可以利用此事件来显示一些剧情，或再让它调用 {“type”: “trigger”} 来继续调用其他的事件。

4.5 开始，难度分歧，获胜与失败

游戏开始时将调用 events.js 中的 startGame 函数。

它将显示 data.js 中的 startText 内容（可以修改成自己的），并正式开始游戏。

其参数 hard 为以下三个字符串之一：”Easy”, “Normal”, “Hard”，分别对应三个难度。针对不同的难度，我们可以设置一些难度分歧。

```
core.hideStartAnimate(function() {
    core.drawText(core.clone(core.firstData.startText), function() {
        core.startGame(hard);
        if (hard=='Easy') { // 简单难度
            core.setFlag('hard', 1); // 可以用flag:hard来获得当前难度
            // 可以在此设置一些初始福利，比如设置初始生命值可以调用：
            // core.setStatus("hp", 10000);
        }
        if (hard=='Normal') { // 普通难度
            core.setFlag('hard', 2); // 可以用flag:hard来获得当前难度
        }
        if (hard=='Hard') { // 困难难度
            core.setFlag('hard', 3); // 可以用flag:hard来获得当前难度
        }
    });
})
```

难度会赋值给 flag:hard，即我们可以在上述 if 语句的 condition 中进行判断。

请不要在任何事件中修改对 flag:hard 进行任何赋值（修改它），不然可能会造成一些不可知的后果。

当获胜（{“type”: “win”}）事件发生时，将调用 events.js 中的 win 事件。其显示一段恭喜文字，并重新开始游戏。

```
////// 游戏结束事件 //////
events.prototype.win = function(reason) {
    // 获胜
    core.waitHeroToStop(function() {
        core.rmGlobalAnimate(0,0,true);
        core.clearMap('all'); // 清空全地图
        core.drawText([
            "\t[结局2]恭喜通关！你的分数是${status:hp}。"
        ], function () {
            core.restart();
        })
    });
}
```

其参数 reason 为获胜原因（即 type:win 事件里面的 reason 参数）。

你可以在这里修改自己的获胜界面显示的文字。

当失败（{“type”: “lose”}，或者被怪强制战斗打死、被领域怪扣血死、中毒导致扣血死，路障导致扣血死等等）事件发生时，将调用 events.js 中的 lose 事件。其直接显示一段文字，并重新开始游戏。

```
events.prototype.lose = function(reason) {
    // 失败
    core.waitHeroToStop(function() {
        core.drawText([
            "\t[结局1]你死了。\n如题。"
        ], function () {
            core.restart();
        });
    })
}
```

其参数 reason 为失败原因。

你可以在这里修改失败界面时显示的文字。

五、 个性化

有时候只靠样板本身可能是不够的。我们需要一些个性化、自定义的素材，道具效果，怪物属性，等等。

5.1 自定义素材

所有素材的图片都在 images 目录下。

animates.png 为所有动画效果。主要是星空熔岩，开门，毒网，传送门之类的效果。为四帧。

enemys.png 为所有怪物的图片。地图生成器中对应的数字，从上至下依次是会从 201 开始计算（即，绿色史莱姆为 201，小蝙蝠为 205，依次类推）。请注意，动画效果为两帧，一般是原始四帧中的 1 和 3。（四帧中 12 相同，34 相同，因此只取 1 和 3 即可）

heros.png 为勇士行走图。这里是 4*3 的，你也可以用 4*4 的，不过需要一些修改，之后会提到。

items.png 为所有道具的图标。

npcs.png 为所有 NPC 的图标，也是两帧。

terrains.png 为所有地形的图标。

系统会读取 icon.js 文件，并获取每个 ID 对应的图标所在的位置。

- **地图生成器使用自定义素材**

地图生成器可以将数字和图标一一对应，从数字生成图标或从图标生成数字。

在使用自定义素材后，我们可以使用地图生成器来识别新的素材。打开同目录下的 meaning.txt，按照已有的方式来增加或编辑内容即可。

第一列是地图生成器中的数字，第二列是它所在的文件名，第三列是坐标。

```
64,items,47 # 绿鞋
65,items,48 # 圣锤
77,items,64 # 你可以随便往后添加，第一列是对应的数字，第二列是图片名，第三列是在图片中的坐标
# 可自行往后添加

### 81-120 门、楼梯、传送门 ###
81,terrains,9 # 黄门
82,terrains,10 # 蓝门
```

- **使用自定义地形（路面、墙壁等）**

你需要自行 P 一张图，里面是你需要的地形素材。然后将其覆盖 terrains.png 文件，请注意所有元件的位置需要完全相同对应。

岩浆、星空、传送门、箭头、门的开启动画、暗墙的开启动画在 animates.png 中，可能也需要对应进行修改。

请打开 items.js 中，对比素材的位置。

另外需要注意的一点是，本样板不支持 Autotile，换句话说野外地图（草地等）这种自然风可能不会太适合。

- **使用自定义道具图标**

如果你觉得已有的道具图标不够，想自己添加图标也是可以的。

如果 items.png 中不存在你需要的图标，则可以自己 P 一张图，将你需要的图标覆盖到某个用不到的图标上。或者也可以接着后面向下拉伸。

所有道具必须是 32x32 像素。

P 图完毕后，可以在地图生成器中加入对应的数字和图标的对应关系。

要在系统中启用你的图标，你需要自己指定一个道具的 ID（不能和任何已有的重名），然后进行如下操作：

1. 在 items.js 中道具的定义列表中编辑，修改你的自定义道具的名称，类型，说明文字等。
    - 即捡即用类道具的 cls 为 items，消耗类道具的 cls 为 tools，永久类道具的 cls 为 constants。
2. 在 icon.js 中，找到 items 一栏，往里面添加你的图标位置。
3. 在 maps.js 中，找到对应位置，往里面添加自己的数字和道具的一一对应关系。（该数字可任意指定，不能和已有的冲突）

```
    if (id == 62) tmp.event = {'cls': 'items', 'id': 'knife'} // 屠龙匕首
    if (id == 63) tmp.event = {'cls': 'items', 'id': 'moneyPocket'} // 金钱袋
    if (id == 64) tmp.event = {'cls': 'items', 'id': 'shoes'} // 绿鞋
    if (id == 65) tmp.event = {'cls': 'items', 'id': 'hammer'} // 圣锤

    // 81-100 门
    if (id == 81) tmp.event = {'cls': 'terrains', 'id': 'yellowDoor', 'trigger': 'openDoor'}; // 黄门
    if (id == 82) tmp.event = {'cls': 'terrains', 'id': 'blueDoor', 'trigger': 'openDoor'}; // 蓝门
```

有关如何自行实现一个道具的效果，参见 5.2。

- **使用自定义怪物图标**

如果你觉得已有的怪物图标不够，也可以自行向后添加你需要的怪物。

在 enemys.js 中，直接向后 P 图，并将你需要的怪物的两帧动画放到正确的位置（注意：普通四帧动画的 1 和 3 帧）。

P 图完毕后，可以在地图生成器中加入对应的数字和图标的对应关系。

要在系统中启用你的图标，你需要自己指定一个怪物的 ID（不能和任何已有的重名），然后进行如下操作：

1. 在 enemys.js 中，向怪物的定义列表中，添加一个怪物。
2. 在 icon.js 中，找到 enemys 一栏，往里面添加你的图标位置。
3. 在 maps.js 中，找到对应位置，往里面添加自己的数字和怪物的一一对应关系。一般对于怪物而言，对应的数字就是 200+位置。

```
    if (id == 255) tmp.event = {'cls': 'enemys', 'id': 'whiteGhost'};
    if (id == 256) tmp.event = {'cls': 'enemys', 'id': 'poisonZombie'};
    if (id == 257) tmp.event = {'cls': 'enemys', 'id': 'magicDragon'};
    if (id == 258) tmp.event = {'cls': 'enemys', 'id': 'octopus'};
    if (id == 259) tmp.event = {'cls': 'enemys', 'id': 'fairy'};
    if (id == 260) tmp.event = {'cls': 'enemys', 'id': 'greenKnight'};

    return tmp;
```

有关如何自行实现一个怪物的特殊属性或伤害计算公式，参见 5.3。

- **使用自定义 NPC 图标**

同上，你也可以自定义 NPC 图标。只需要将你 NPC 的两帧动画接到 npcs.js 中即可。

P 图完毕后，可以在地图生成器中加入对应的数字和图标的对应关系。

要在系统中启用你的图标，你需要自己指定一个 NPC 的 ID（不能和任何已有的重名），然后进行如下操作：

1. 在 icon.js 中，找到 npcs 一栏，往里面添加你的图标位置。
2. 在 maps.js 中，找到对应位置，往里面添加自己的数字和 NPC 的一一对应关系。

```

    // 121-150 NPC
    if (id == 121) tmp.event = {'cls': 'npcs', 'id': 'man'};
    if (id == 122) tmp.event = {'cls': 'npcs', 'id': 'woman'};
    if (id == 123) tmp.event = {'cls': 'npcs', 'id': 'thief'};
    if (id == 124) tmp.event = {'cls': 'npcs', 'id': 'fairy'};
    if (id == 125) tmp.event = {'cls': 'npcs', 'id': 'magician'};
```

- **使用自定义勇士图标**

我们同样可以使用自定义的勇士图标，比如小可绒之类。

拿一个勇士的行走图覆盖 hero.png 即可。

请注意：勇士必须是 32x32 像素，不能使用超过 32x32 的图片。

覆盖了 hero.png 后，我们需要在 icons.js 中编辑图标的位置信息。

```
'heros': {
    'hero1': {
        'down': {'loc': 0, 'stop': 0, 'leftFoot': 1, 'rightFoot': 2},
        'left': {'loc': 1, 'stop': 0, 'leftFoot': 1, 'rightFoot': 2},
        'right': {'loc': 2, 'stop': 0, 'leftFoot': 1, 'rightFoot': 2},
        'up': {'loc': 3, 'stop': 0, 'leftFoot': 1, 'rightFoot': 2}
    }
},
```

hero1 为勇士 ID，和 data.js 中的 ID 对应，一般不修改。

接着 down/left/right/up 是勇士四个朝向，每个朝向中的 loc 表示是在 hero.png 中的第几行；后面的 stop，leftFoor，rightFoot 分别是停止图，左脚行走图和右脚行走图在该行的第几列就行。

我们只需要覆盖图片后编辑这里即可。即使是 4x4 的行走图，也是没问题的（需要指定左脚和右脚所在的第几列）。

通过上述这几种方式，我们可以修改素材图片（使用自定义素材），指定数字并放入地图生成器中，然后在系统中进行启用。

请注意：**强制要求所有素材都必须是 32x32 的**，不然可能会造成不可预料的后果。

5.2 自定义道具效果

本节中将继续介绍如何自己编辑一个道具的效果。

道具效果的具体实现都在 items.js 中。

- **即捡即用类道具（cls: items）**

对于即捡即用类道具，如宝石、血瓶、剑盾等，我们可以简单地修改 data.js 中的 value 一栏即可。

如果你有更高级的需求（例如每个区域的效果不同），则需要编辑 items.js 文件。具体方式是：

1. 找到 getItemEffect 函数；所有即捡即用类道具的效果都在这里实现。
2. 计算道具效果系数，或应该增加的值。

```
    // 获得当前楼层：name和剧本文件中的name完全一致
    var floor = parseInt(core.status.thisMap.name);
    var ratio = 1; // 道具效果系数
    if (floor>=11 && floor<=20) ratio=2; // 二区翻倍
    if (floor>=21 && floor<=30) ratio=3; // 倍效果，也可以四倍等
    // ... 根据自己的需要

    // 具体获得道具的效果可以进行相应的修改（乘以系数）
    if (itemId === 'redJewel') core.status.hero.atk += core.values.redJewel * ratio;
    if (itemId === 'blueJewel') core.status.hero.def += core.values.blueJewel * ratio;
    if (itemId === 'greenJewel') core.status.hero.mdef += core.values.greenJewel * ratio;
    if (itemId == 'yellowJewel') { // 黄宝石属性：需自己定义
```

请注意这里 core.status.thisMap.name 获取的是当前层中，你在剧本文件里写的 name 那一项（即状态栏中的层数显示）。然后可以通过几个简单的 if 来判断应该增加的值。

3. 修改同样修改下面的 getItemEffectTip 函数，使提示文字相应变动。

- **消耗类道具（cls: tools）；永久类道具（cls: constants）**

如果要自己实现消耗类道具或永久类道具的使用效果，则需修改 items.js 中的 canUseItem 和 useItem 两个函数。

具体过程比较复杂，需要一定的 JS 能力，在这里就不多说了，有需求可以找艾之葵进行了解。

但值得一提的是，我们可以使用 core.hasItem(name) 来判断是否某个道具是否存在。例如下面是 passNet（通过路障处理）的一部分：

```
/****** 经过路障 ******/
events.prototype.passNet = function (data) {
    // 有鞋子
    if (core.hasItem('shoes')) return;
    if (data.event.id=='lavaNet') { // 血网
        core.status.hero.hp -= core.values.lavaDamage;
        if (core.status.hero.hp<=0) {
```

我们进行了一个简单的判断，如果拥有绿鞋，则不进行任何路障的处理。

- **实战！拿到神圣盾后免疫吸血、领域、夹击效果**

1. 在 getItemEffect 中修改拿到神圣盾时的效果，标记一个自定义 Flag。

```
    if (itemId === 'shield5') { // 拿到神圣盾后
        core.status.hero.def += core.values.shield5;
        core.setFlag("hasShield5", true); // 设置一个自定义flag，表示已经有了神圣盾。
    }
```

2. 免疫吸血效果：在 enemys.js 的 getDamage 函数中，找到 extra_damage，并编辑成如果存在神圣盾标记，额外伤害为 0。

```
var extra_damage = 0;
if (monster.special == 11) { // 吸血
    // 吸血的比例
    extra_damage = core.status.hero.hp * monster.value;
    if (core.hasFlag("hasShield5")) // 如果存在神圣盾标记，则没有额外伤害
        extra_damage = 0;
    extra_damage = parseInt(extra_damage);
}
return damage + extra_damage;
```

3. 免疫领域、夹击效果：在 events.js 中，找到 checkBlock 函数，并编辑成如果有神圣盾标记，则直接返回。

```
////// 检查领域、夹击事件 //////
events.prototype.checkBlock = function (x,y) {
    if (core.hasFlag('hasShield5')) // 如果存在神圣盾，不进行检查，直接返回
        return;
    var damage = 0;
    // 获得四个方向的怪物
    var directions = [[0,-1],[-1,0],[0,1],[1,0]]; // 上，左，下，右
```

4. 如果有更高的需求，例如想让吸血效果变成一半（如异空间），则还是在上面这些地方进行对应的修改即可。

5.3 自定义怪物属性和伤害计算公式

如果你对现有的怪物不满意，想自行添加怪物属性（例如让怪物拥有双属性乃至更多属性），也是可以的。具体参见 enemys.js 文件。

你需自己指定一个 special 数字，修改 getSpecialText 函数。

如果要修改伤害计算公式，请修改下面的 calDamage 函数。请注意，如果无法战斗，该函数必须返回 999999999。

因此无敌属性可以这样设置：

```
enemys.prototype.calDamage = function (hero_atk, hero_def, hero_mdef, mon_hp, mon_atk, mon_def, mon_
    // 增添无敌属性
    // 假设设置special数值为18
    if (mon_special==18 && !core.hasItem('cross')) // 如果是无敌属性，且不存在十字架
        return 999999999; // 直接返回999999999代表不可战胜

    // 魔攻
```

对于吸血怪的额外伤害计算在 getDamage 中的 extra_damage 中。

对于毒衰弱怪物的战斗后结算在 events.js 中的 afterBattle 函数中。

对于领域、夹击怪物的检查在 events.js 中的 checkBlock 函数中。

getCritical, getCriticalDamage 和 getDefDamage 三个函数依次计算的是该怪物的临界值、临界减伤和 1 防减伤。也可以适当进行修改。

5.4 自定义地图

遗憾的是，所有地图数据必须在剧本的 map 中指定，换句话说，我们无法在游戏进行中动态修改地图，比如为简单难度增加一个血瓶。

幸运的是，我们可以采用如下方式进行难度分歧，为用户简单难度下增加额外的血瓶或宝石。

```
],
"firstArrive": [ // 第一次到该楼层触发的事件
    {"type": "if", "condition": "flag:hard!=3", // 条件判断：是否困难难度
        "true": [ // 不为困难难度，则为普通货简单
            {"type": "show", "loc": [3,6]} // 显示血瓶
            {"type": "if", "condition": "flag:hard==1", // 再次条件判断：是否为简单难度
                "true": [{"type": "show", "loc": [3,7]}], // 如果是，则显示宝石
                "false": [] // 普通难度，只显示了血瓶
            }
        ],
        "false": [] // 如果是困难难度，不进行任何操作
    }
],
"events": { // 该楼的所有可能事件列表
    "3,6": {"enable": false} // 比如[3,6]点是一个血瓶，初始时不可见
    "3,7": {"enable": false} // [3,7]是一个宝石，初始时不可见
```

如图所示，我们在地图上设置一个额外的血瓶和宝石，并初始时设为禁用状态。

当第一次到达该楼层时，进行一次判断；如果不为困难难度，则将血瓶显示出来；再判断是否为简单难度，如果是则再把宝石显示出来。

通过对 flag:hard 进行判断的方式，我们也可以达成“对于不同的难度有着不同的地图效果”。

附录：所有 API 列表

所有系统支持的 API 都列在了这里。可能被做塔时自定义 JS 脚本中涉及到的以红色字体标出，并有着详细的解释。

可以在 chrome 浏览器的控制台中（ctrl+shift+I，找到 Console）中直接进行调用，以查看效果。

core.js：系统核心文件。所有核心逻辑处理都在此文件完成。

core.status.floorId 获得当前层 floorId
core.status.thisMap 获得当前层的地图信息
------ **初始化部分** ------
core.init 初始化
core.showStartAnimate 显示开始界面
core.hideStartAnimate 隐藏开始界面
core.setStartProgressVal 设置加载进度条进度
core.setStartLoadTipText 设置加载进度条提示文字
core.loader 加载图片和音频
core.loadImage 加载图片
core.loadSound 加载音频
core.loadSoundItem 加载某一个音频
core.isPlaying 游戏是否已经开始
core.clearStatus 清除游戏状态和数据
core.resetStatus 重置游戏状态和初始数据
core.startGame 具体开始游戏
core.restart 重新开始游戏；此函数将回到标题页面
------ **键盘、鼠标事件** ------
core.onKeyDown 按下某个键时
core.onKeyUp 放开某个键时
core.pressKey 按住某个键不动时
core.keyDown 根据按下键的 code 来执行一系列操作
core.keyUp 根据放开键的 code 来执行一系列操作
core.ondown 点击（触摸）事件按下时
core.onmove 当在触摸屏上滑动时
core.onup 当点击（触摸）事件放开时
core.getClickLoc 获得点击事件相对左上角的坐标（0 到 12 之间）

core.onclick 具体点击屏幕上(x,y)点时，执行的操作
core.onmousewheel 滑动鼠标滚轮时的操作（楼层传送时可用滚轮切换楼层）
------ 自动寻路代码相关 ------
core.clearAutomaticRouteNode 清除自动寻路路线
core.stopAutomaticRoute 停止自动寻路操作
core.continueAutomaticRoute 继续剩下的自动寻路操作
core.clearContinueAutomaticRoute 清除剩下的自动寻路列表
core.setAutomaticRoute 设置一个自动寻路
core.automaticRoute 自动寻路算法，找寻最优路径
core.fillPosWithPoint 显示离散的寻路点
core.clearStepPostfix 清除已经寻路过的部分
------ 自动行走，行走控制 ------
core.stopAutoHeroMove 停止勇士的自动行走
core.setAutoHeroMove 设置勇士的自动行走路线
core.autoHeroMove 让勇士开始自动行走
core.setHeroMoveInterval 设置行走的效果动画
core.setHeroMoveTriggerInterval 设置勇士行走过程中对途经事件的触发检测
core.turnHero(direction) 设置勇士的方向（转向）；如果指定了 direction 则会面向该方向，否则执行一个转向操作。
core.moveHero 让勇士开始移动
core.moveOneStep 每移动一格后执行的事件。中毒时在这里进行扣血判断。
core.waitHeroToStop(callback) 停止勇士的一切行动，等待勇士行动结束后，再执行 callback 回调函数。
core.stopHero 停止勇士的移动状态。
core.drawHero 在 hero 层绘制勇士。
core.setHeroLoc(name, value) 设置勇士的位置。name 为”direction”,”x”,”y”
core.getHeroLoc(name) 获得勇士的位置。
core.nextX 获得勇士面对位置的 x 坐标
core.nextY 获得勇士面对位置的 y 坐标
------ 地图和事件处理 ------
core.openDoor(id, x, y, needKey, callback) 打开一扇位于 (x,y) 的门
core.battle(id, x, y, force, callback) 进行战斗；force 表示是否强制战斗
core.trigger(x,y) 触发 x,y 点的事件
core.changeFloor(floorId, stair, heroLoc, time, callback) 楼层切换
floorId 为目标楼层 Id，stair 可指定为上/下楼梯，time 动画时间
core.mapChangeAnimate 实际切换的动画效果

core.clearMap 清除地图显示
core.fillText 在某个 canvas 上绘制一段文字
core.fillRect 在某个 canvas 上绘制一个矩形
core.strokeRect 在某个 canvas 上绘制一个矩形的边框
core.setFont 设置某个 canvas 的文字字体
core.setLineWidth 设置某个 canvas 的线宽度
core.saveCanvas 保存某个 canvas 状态
core.loadCanvas 读取某个 canvas 状态
core.setStrokeStyle 设置某个 canvas 边框属性
core.setAlpha 设置某个 canvas 的 alpha 值
core.setOpacity 设置某个 canvas 的透明度
core.setFillStyle 设置某个 canvas 的绘制属性（如颜色等）
core.drawMap(mapId, callback) 绘制某张地图。mapId 为地图 Id，绘制完毕将执行 callback 回调函数。
core.noPassExists(x,y) 某个点是否不可通行
core.noPass 某个点是否在区域内且不可通行
core.npcExists(x,y) 某个点是否存在 NPC
core.terrainExists(x,y) 某个点是否存在指定的地形
core.stairExists(x,y) 某个点是否存在楼梯
core.nearStair 当前位置是否在楼梯边
core.enemyExists(x,y) 某个点是否存在怪物
core.getBlock(x, y, floorId, needEnable) 获得某个点的 block。
floorId 指定目标楼层，needEnable 如果为 false 则即使该点的事件处于禁用状态也将被返回（否则只有事件启用的点才被返回）
core.moveBlock 显示移动某块的动画，达到{“type”:”move”}的效果
core.animateBlock 显示/隐藏某个块时的动画效果
core.addBlock 将某个块从禁用变成启用状态
core.removeBlock 将某个块从启用变成禁用状态
core.removeBlockById 根据 block 的索引删除该块
core.removeBlockByIds 一次性删除多个 block
core.addGlobalAnimate 添加一个全局动画
core.removeGlobalAnimate 删除一个或所有全局动画
core.setGlobalAnimate 设置全局动画的显示效果
core.setBoxAnimate 显示 UI 层某个 box 的动画（如怪物手册中怪物的动画）
core.drawBoxAnimate 绘制 UI 层的 box 动画
core.updateFg() 更新全地图的显伤

core.itemCount 获得某个物品的个数
core.hasItem 是否存在某个物品
core.setItem 设置某个物品的个数
core.removeItem 删除某个物品
core.useItem 使用某个物品；直接调用 items.js 中的 useItem 函数。
core.canUseItem 能否使用某个物品。直接调用 items.js 中的 canUseItem 函数。
core.addItem 增加某个物品的个数
core.getItem 获得某个物品时的事件
core.drawTip 左上角绘制一段提示
core.drawText 地图中间绘制一段文字
------ **系统机制** ------
core.replaceText 将文字中的${和}（表达式）进行替换
core.calValue 计算表达式的值
core.unshift 向某个数组前插入另一个数组或元素
core.setLocalStorage 设置本地存储
core.getLocalStorage 获得本地存储
core.removeLocalStorage 移除本地存储
core.clone 复制一个对象
core.formatDate 格式化时间为字符串
core.setTwoDigits 两位数显示
core.win 获胜；将直接调用 events.js 中的 win 函数
core.lose 失败；将直接调用 events.js 中的 lose 函数
core.debug 进入 Debug 模式，攻防血和钥匙都调成很高的数值
core.checkStatus 判断当前能否进入某个事件
core.openBook 点击怪物手册时的打开操作
core.useFly 点击楼层传送器时的打开操作
core.openToolbox 点击工具栏时的打开操作
core.save 点击保存按钮时的打开操作
core.load 点击读取按钮时的打开操作
core.doSL 实际进行存读档事件
core.syncSave 存档同步操作
core.saveData 存档到本地
core.loadData 从本地读档
core.setStatus 设置勇士属性
core.getStatus 获得勇士属性
core.setFlag 设置某个自定义变量或 flag

core.getFlag 获得某个自定义变量或 flag
core.hasFlag 是否存在某个自定义变量或 flag，且值为 true
core.insertAction 往当前事件列表之前插入一系列事件
core.lockControl 锁定状态栏，常常用于事件处理
core.unlockControl 解锁状态栏
core.isset 判断某对象是否不为 undefined 也不会 null
core.playSound 播放音频
core.playBgm 播放背景音乐
core.changeSoundStatus 切换声音状态
core.enableSound 启用音效
core.disableSound 禁用音效
core.show 动画显示某对象
core.hide 动画使某对象消失
core.clearStatusBar 清空状态栏
core.updateStatusBar 更新状态栏
core.resize 屏幕分辨率改变后重新自适应
core.resetSize 屏幕分辨率改变后重新自适应
------ **core.js 结束** ------

data.js 定义了一些初始化的数据信息。

enemys.js 定义了怪物信息。

core.enemys.getSpecialText 获得特殊属性的文字
core.enemys.getDamage 获得某个怪物的伤害
core.enemys.getCritical 计算某个怪物的临界值
core.enemys.getCriticalDamage 计算某个怪物的临界减伤
core.enemys.getDefDamage 计算某个怪物的 1 防减伤
core.enemys.calDamage 实际的伤害计算公式
core.enemys.getCurrentEnemys 获得当前层剩下的的怪物列表

events.js 定义了各个事件的处理流程。

core.events.startGame 开始游戏
core.events.win 获胜
core.events.lose 失败
core.events.checkBlock 检查领域、夹击事件
core.events.afterChangeFloor 楼层切换结束时的事件
core.events.doEvents 开始执行一系列自定义事件
core.events.doAction 执行当前自定义事件列表中的下一个事件
core.events.insertAction 往当前自定义事件列表前插入若干个事件
core.events.openShop 打开一个全局商店
core.events.disableQuickShop 禁用一个快捷商店
core.events.canUseQuickShop 当前能否使用快捷商店
core.events.useItem 尝试使用道具
core.events.afterBattle 战斗结束后触发的事件
core.events.afterOpenDoor 开一个门后触发的事件
core.events.passNet 经过一个路障
core.events.beforeSaveData 即将存档前可以执行的操作
core.events.afterLoadData 读档后，载入事件前可以执行的操作
------ **界面上的点击事件** ------
core.events.clickAction 自定义事件处理时，对用户点击的处理
core.events.clickBook 怪物手册打开时，对用户点击的处理
core.events.clickFly 楼层传送器打开时，对用户点击的处理
core.events.clickShop 全局商店打开时，对用户点击的处理
core.events.clickQuickShop 快捷商店选项打开时
core.events.clickToolbox 工具栏打开时
core.events.clickSL 存/读档界面打开时
core.events.clickSettings 设置页面打开时

icons.js 定义了各个图标所在的文件、位置。

items.js 定义了各个道具的效果。

core.items.getItemEffect 获得某个即捡即用道具时的效果
core.items.getItemEffectTip 获得某个即捡即用道具时的提示文字
core.items.useItem 实际使用某个道具
core.items.canUseItem 当前能否使用某个道具

maps.js 定义了地图，以及每个数字所代表的意义。

core.maps.loadFloor 加载某个楼层（从剧本或存档中）
core.maps.getBlock 将数字替换成实际的内容
core.maps.addEvent 向该楼层添加剧本的自定义事件
core.maps.addChangeFloor 向该楼层添加剧本的楼层转换事件
core.maps.initMaps 初始化所有地图
core.maps.save 将当前地图重新变成数字，以便于存档
core.maps.load 将存档中的地图信息重新读取出来

ui.js 定义了各种界面的绘制。

core.ui.closePanel 结束一切事件和绘制，关闭 UI 窗口，返回游戏进程
core.ui.drawTextBox 绘制一个对话框
core.ui.drawChoices 绘制一个选项界面
core.ui.drawConfirmBox 绘制一个确认/取消的警告页面
core.ui.drawSettings 绘制系统菜单栏
core.ui.drawQuickShop 绘制快捷商店选择栏
core.ui.drawWaiting 绘制一个“请稍后”页面
core.ui.drawSyncSave 绘制存档同步选项
core.ui.drawPagination 绘制分页
core.ui.drawEnemyBook 绘制怪物手册
core.ui.drawFly 绘制楼层传送器
core.ui.drawToolbox 绘制道具栏
core.ui.drawSLPanel 绘制存档/读档界面
core.ui.drawThumbnail 绘制一个缩略图
core.ui.drawAbout 绘制“关于”界面