PIX37 使用手册

先编辑，再编写代码。本书的离线版本可在 pix37.party/docs 上找到。

编辑

PIX37 DEV 是编辑器。您可以在这里绘制、编写和播放游戏，然后导出游戏卡带或独立版本。它还可以运行游戏卡带，并可打开 PIX37.PARTY 下载游戏卡带和使用游戏目录。PIX37 DEV EDUCATIONAL 是相同的编辑器，并且是免费的。它不提供游戏版本、游戏卡带导出功能，也无法访问 PIX37.PARTY。PIX37 Console 是播放器。它可以运行 `.p37` 游戏卡带，您可以通过打开 PIX37.PARTY 下载游戏卡带和浏览游戏目录。

项目和游戏卡带

`.pix37` 文件是项目文件：包含源文件、精灵图、地图、声音和数据表。`.p37` 文件是玩家运行的游戏卡带：包含紧密打包的字节码。

存档和成就文件位于应用程序数据文件夹旁边的作者文件夹中。在 Windows 系统中，存档文件类似于 `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s`，成就文件类似于 `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach`。在 Linux 系统中，对应的文件分别是 `~/.local/share/PIX37/Mira/Pong.p37s` 和 `~/.local/share/PIX37/Mira/Pong.ach`。Ada 是计算机上的账户名称。Mira 是作者名称，也是文件夹名称。Pong 是存档槽位，默认为游戏标题。`save` 会写入 `.p37s`。第一个解锁的成就会写入 `.ach`。如果这些文件已经存在，程序会在游戏卡带启动时读取它们。

开发者菜单

按 Esc 键打开开发者菜单。游戏运行时按 Esc 键返回编辑器。按 Enter 键或 A 键（默认为 V 键）确认。返回键是 B 键（默认为 C 键）。

如果没有加载任何项目，菜单为：创建项目、加载项目、构建、加载游戏卡带、连接到 PIX37.PARTY、设置。

加载项目后，可执行以下操作：保存项目、加载项目、创建项目、资源、构建、加载游戏卡带、连接到 PIX37.PARTY、设置。

按Ctrl+S保存打开的项目。按Ctrl+R运行它。

创建项目

创建项目时，需要先插入游戏卡带才能开始制作项目。

- **视频**——1 位（2 种颜色）、2 位（4 种颜色）、4 位（16 种颜色）、8 位（256 种颜色）。
- **音频**— 合成器的采样深度：16、8、4、2 或 1 位。

- **屏幕**——预设尺寸包括 64×64、128×128、160×90、256×256 和 320×180。自定义尺寸允许您选择 1×1 到 512×512 之间的任何尺寸。
- **调色板**——颜色库。第一个颜色库包含以下 16 种颜色。加载调色板会应用一个内置颜色集：PICO-8 16、PICO-8 32、Game Boy、1 位、CGA、Sweetie-16，以及来自 Lospec 的更长列表（Resurrect 64、Apollo、Endesga 32 和 64、CC-29、SLSO8、Pear36、Oil 6、Duel、AAP-64、Lost Century 等）。您还可以加载调色板文件，或者加载您自己的 PNG 图片并从中提取调色板。



`cls`、`pset`、`spr` 等颜色索引均已设置。您可以在调色板编辑器中重新绘制任何索引。

五个面板

编辑器依次为 GAME、PAINT、CODE、SOUND 和 DATA。地图位于 PAINT 面板中。音效和歌曲位于 SOUND 面板中。点击面板即可进行编辑。当 CODE 面板获得焦点时，SOUND 和 DATA 面板会忽略字母键，因此输入文字不会同时弹出钢琴声。



完整布局，格式为 `1:1:4:1:1`。每个数字表示该面板在行中所占的比例：游戏、绘画、代码、声音、数据。一个小屏幕位于 128×128 像素单元格的中央。屏幕尺寸大于 128 像素时，游戏列的宽度会增加。

设置 → 全局 → 布局，循环切换布局：

代码	你所看到的
`1:1:4:1:1`	一切尽在其中。游戏和声音堆叠在一起，绘图和数据堆叠在一起，代码在右侧。
`1:1:4:1:1 2X`	布局相同。代码的绘制密度是原来的两倍，因此可以容纳两倍的文本。
`1:0:1:0:1`	游戏、数据、代码。
`1:0:1:0:1 2X`	同样的布局，但代码更密集。
`0:1:1:1:1`	无游戏预览。绘图、代码、声音、数据。
`0:1:1:1:1 2X`	同样的布局，但代码更密集。
`0:1:4:1:2`	工作室。宽大的 CODE 插槽、更高的 DATA 插槽、PAINT 和 SOUND 插槽。
`0:1:4:1:2 2X`	同样的布局，但代码更密集。
`0:0:4:0:4`	仅包含代码和数据，两者均很大。
`0:0:4:0:4 2X`	同样的布局，但代码更密集。
`0:1:4:1:0`	绘画、声音和广泛的代码。
`0:1:4:1:0 2X`	同样的布局，但代码更密集。
`0:4:0:0:0`	仅绘画，256×256，纸张上的一个像素对应画布上的一个像素。

`2X` 行保留相同的面板。代码面板以两倍分辨率绘制，并缩放回同一单元格，因此文本、制表符和间距的高度减半，屏幕上可显示两倍的源代码。

游戏

预览正在运行的游戏卡带。点击"游戏"面板，或按 Ctrl+R 即可运行预览。面板配置位于"设置"→"游戏面板"中：

- **预览**——空闲颜色、连续或编辑时自动重新加载。
- **在新窗口中运行**——游戏会在编辑器旁边打开。
- **显示**— 预览为 RGB888 或 RGB565。
- **游戏 64** — 64×64 屏幕要么以 1:1 的比例居中显示在单元格中，要么缩放 2 倍以填充单元格。
- **着色器**——游戏中的 CRT 和滤镜，与编辑器中的滤镜是分开的。
- **颜色**——面板的背景色。

画

精灵图，以及位于同一面板地图侧的瓦片地图。图块可以大于屏幕。精灵预览尺寸为 8、16、32、64 和 128（仅当图块至少为 128 像素时才支持 128）。在精灵预览上滚动鼠标滚轮即可缩放。笔刷大小为 1、2、3 和 4。

钥匙	工具
B	铅笔
M	矩形选择
问	套索
S	邮票
G	桶
我	滴管
V	移动
L / Shift+L	线/曲线
U / Shift+U	空心矩形/空心椭圆
F / Shift+F	填充矩形/填充椭圆
D	抖动
[ ]	小号/大号刷子
Ctrl+Z / Ctrl+Y	撤销/重做。Ctrl+Shift+Z 也可重做。
Ctrl+C / Ctrl+X / Ctrl+V	复制、剪切、粘贴
Ctrl+A	选择画布

地图的设计思路相同，但工具较少：B 铅笔，M 矩形选择，S 图章，V 移动，以及 Ctrl+Z、Ctrl+Y、Ctrl+C、Ctrl+V 快捷键。按住 Alt 键可缩放地图纹理。地图上的笔刷大小与绘图工具保持一致。

每个精灵都有八个标志位。绘制程序会将这些标志位设置到精灵上；`fget` 和 `fset` 会在游戏中读取这些标志位。

该面板的配置位置在“设置”→“绘图编辑器”中：

- **调色板**——选择哪个银行，以及加载和保存。
- **预览**— RGB888 或 RGB565。
- **画笔光标**——绘画时的指针。
- **像素捕捉**——画笔如何捕捉到像素。
- **大刷子**——一个大刷子从它的左上角或整个足迹处弹出。

- **洋葱皮**——其他帧的幽灵，以及该幽灵面向的方向。
- **地图缩放**——真实像素缩放或自由缩放。
- **颜色**——用户界面颜色。

代码

选项卡是项目中的独立文件。单击选项卡名称即可重命名；按 Enter 键确认。定义 `_init`、`_update` 或 `_draw` 的多个选项卡会按选项卡顺序拼接在一起：第一个选项卡最先运行，最后一个选项卡最后运行。不在钩子内的代码会成为 `_init` 的一部分。完全没有钩子的游戏卡带会被视为一个 `_draw`。

钥匙	行动
箭头	移动光标。按住Shift键可扩大选择范围。
Ctrl+左/右	用言语。
Home / End，Ctrl+Home / End	行首或行尾，或制表符的开头或结尾。
标签页	插入两个空格。在补全列表中，接受该名称。
进入	另起一行，缩进方式与上一行相同。接受完成操作。
Ctrl+C / X / V	复制、剪切、粘贴。
Ctrl+Z / Ctrl+Y	撤销/重做。
Ctrl+A	选择标签页。
Ctrl+D	向下复制该行或选区。
Ctrl+/	使用 `--` 进行注释或取消注释。
Ctrl+F	查找。F3 跳转到下一个目标。Esc 键关闭工具栏。
Ctrl+G	跳转到指定行号。
逃脱	首先关闭查找、跳转或提示列表。再次按下 Esc 键将退出编辑器，返回开发者菜单。

输入函数名时会弹出提示。按 Enter 或 Tab 键可插入函数名，如果"完全"功能已在设置中启用，则会显示 `(`。设置 → 代码编辑器：

- 在当前标签页或所有标签页中按 Ctrl+F**搜索**。
- **文本高度**— 将字形裁剪为 5 像素，或显示完整字形。
- **字符串大小**——字符串字面量的字体大小。
- **注释字体**— 3×5，或当前 UI 语言的 Fusion 字体。

- **完成（** —是否完成括号。
- **代码提示**——开启或关闭。
- **提示语言**——菜单语言，即英语。
- **插入符空格**——空行上的额外偏移量。
- **页边线**——行号。
- **颜色**— 背景、注释、制表符、数字、运算符、关键字、字符串、引擎函数、您的函数、变量、括号、插入符、选择、错误行、警告。

声音

面板有两个选项卡。“SFX”代表一种音型：一小段音符。“SOUND”代表一首歌曲：它将这些音型分配到八个通道上。“TIC”表示一个音符步长所对应的节拍数。时钟每秒 120 个节拍，一个音符步长为十六分音符，因此 15 个节拍即为每分钟 120 拍。TIC 值越高，音型速度越慢。当一个 TIC 值能够整除另一个 TIC 值时，两个音型保持同步：例如，15 和 30 可以，但 15 和 8 则不能。

用鼠标选中音符即可演奏钢琴。白键为 ZXCVBNM（CDEFGAB）。黑键为 SDGHJ（C# D# F# G# A#）。将鼠标悬停在任意数值上并输入数字，或将鼠标悬停在数值上并滚动鼠标滚轮。代码中的音高编号范围为 0（C-0）到 63（D#5）。

在一种乐句中，“LOOP”和“LEN”是相同的两个数字。当“OUT”大于0时，乐句从“IN”循环到“OUT”，标题显示为“LOOP”。当“OUT”为0且“IN”大于0时，“LEN”表示乐句从头开始播放相应数量的音符，然后乐句结束，标题显示为“LEN”。当“OUT”和“LEN”均为0时，乐句会播放每个音符。

你能编辑多少合成器音色取决于游戏卡带选择的位深度。“设置”→“音效编辑器”可以设置位深度、音符是否自动前进（关闭、音符链或场链）、空格键是重新开始播放还是切换开始/停止，以及编辑器颜色。

16 位

每个步骤都包含音符、音量、波形和效果。音量范围为 0 到 7，0 为静音。

波浪：

- **0 三角形**
- **1 倾斜锯**
- **2 锯**
- **3 平方**
- **4 脉冲**——窄脉冲
- **5个器官**
- **6 噪音**
- **7相**
- **8 PWM——**一个宽度移动的正方形
- **9 脉冲 25%**
- **10 正弦**

- **11 金属噪音**
- **12 FM 铃声**
- **13 拔毛**
- **14 超级锯**
- **15 PWM（C64）**

单音符效果：

- **0 无**
- **1 滑音**— 音高和音量滑到此步骤的下一个音符
- **2. 颤音**——音调波动
- **3 降调**——音调降至静音，通常的踢腿声
- **4.淡入**
- **5 淡出**
- **6 琶音**——每 8 个节拍演奏一个音符，然后演奏接下来的三个音符。在节拍 8 或以下时，每 4 个节拍演奏一个音符。
- **7 快速琶音**——相同的四个音符，每 4 个节拍演奏一次，或者当 TIC 为 8 或更低时，每 2 个节拍演奏一次。

图案标题（位于注释上方）：

- **TIC——**每步的节拍数，从 1 到 255。15 表示每分钟 120 次节拍。输入数字，或每次加 1。
- **循环输入/输出**— 当 OUT 大于 0 时，乐段会从输入循环到输出。当 OUT 为 0 时，标题变为 LEN：输入表示播放的音符数量，0 表示整个乐段。
- **噪声**——整个图案都变成了噪声。
- **CRUSH——**每款浪潮的活力变种。
- **DTUNE** — 0 关闭，1 轻柔合唱，2 强烈合唱。
- **RVERB** — 混响：0 关闭，1 短，2 长。
- **DAMPN** — 低通滤波器：0 关闭，1 弱，2 强。
- **级别**— 整个图案的音量，0 到 7。
- **摇摆步**——小碎步。奇数步稍稍延后，偶数步稍稍延后。
- **步骤**— 该图案使用多少步：32、24、16 或 12。

8 位

一个步进包含一个音符和一个波形。四种波形分别是方波、三角波、锯齿波和噪声波。头部包含 TIC、LOOP IN / OUT、FX 和 LEVEL。TIC 的值可以是 1、2、3、4、5、6、8、10、12、15、16、20、24、30、40 或 60。FX 是整个乐段的单一效果：0 无效果，1 颤音，2 琶音，3 淡出。LEVEL 的值范围为 0 到 3。

4 位

一个音符就是一个音阶。标题栏包含 TIC、LOOP（开/关）、WAVE（四种波形）和 OCT（音符所在的八度）。TIC 的值可以是 4、6、8、10、12、15、20 或 30。

2 位

一个音阶可以是空的、鼓点、重击或重击。鼓头有 TIC 和 BASE 两个参数。TIC 的取值可以是 8、15、16 或 30。BASE 设置鼓点和重击的高度。

1 位

一步即一个音符。TIC 为 8、15、16 或 30，与 2 位数字的四种速度相同。

歌曲

"声音"选项卡显示的是当前播放的歌曲。八列代表不同的声道。每个单元格代表一个模式 ID。例如，单元格 255 为空，表示该声道在该行静音。X 列是命令：继续、停止、循环开始或循环结束。停止命令会在当前行结束后结束歌曲。循环结束命令会跳回上一个循环开始的位置。

当最左侧的 LOOP OUT 位于 0 的花样完成时，一行结束。更右侧的花样不会延长该行：较长的花样会被剪掉，较短的花样会从第一步重新开始，直到该行结束。LOOP OUT 位于 0 以上的花样会在 LOOP IN 和 LOOP OUT 之间循环，不会结束该行。如果该行中的所有花样都循环，则该行保持不变。

快捷方式

钥匙	行动
钢琴键	在用鼠标选择的步骤上写下备注。
上/下	更改光标下的值。
空间	播放第一行的起始音效或起始歌曲。可在音效编辑器设置中开始/停止或重新播放。
Ctrl+Z / Ctrl+Y	撤销/重做。
Ctrl+C / Ctrl+X / Ctrl+V	复制、剪切、粘贴所选内容。

数据

命名表。每个表都包含行和列。代码中的 `dget`、`dset`、`dstr`、`drows` 和 `dcols` 使用了这些表。代码中的行号和列号均从 0 开始。

列类型，按标题循环切换：

- **布尔值**——一位。
- **2——**两比特。
- **4** — 四比特。

- **8**——一个字节。
- **16** — 16 位，用于 ID 和更大的数字。
- **spr** — 精灵 ID。
- **str** — 文本，使用系统字体绘制。
- **str8**和**str10** — 相同的文本，以更高的形式显示。
- **f16**和**f32** — 半精度浮子和全浮子。

将指针放在列标题上，按住 Ctrl 键并滚动鼠标滚轮即可更改该列的类型。普通的滚动条可以横向或纵向移动，具体取决于"设置"→"数据编辑器"中的设置。Ctrl 键可以切换滚动方向，这就是为什么当滚动条设置为"列"时，按住 Ctrl 键并滚动标题可以更改列的类型。

方向键用于移动。回车键用于编辑单元格。设置 → 数据编辑器：

- **滚轮**——滚动列或滚动行。
- **颜色**——网格、标题和选区。

资产

ASSETS 导入和导出的是当前打开的素材，而不是整个项目。Sprites 使用的是在 PAINT 中打开的、位于当前打开的绘图板上的精灵。map 指的是当前打开的地图。SFX 指的是当前打开的音效。Music 指的是当前打开的歌曲。Data 指的是当前打开的表格。Code 是个例外：每个标签页都会被写入，每个标签页也会被读取。

种类	文件	什么在移动
精灵	`.png`	打开精灵。
地图	`.p37map`	开放地图。
音效	`.p37sfx`	开放式图案。
音乐	`.p37mus`	开头的歌曲。模式编号保持不变。如果需要音效，请单独导出。
数据	`.p37dat`	敞开的桌子。
代码	`.lua`	每个标签页。

设置

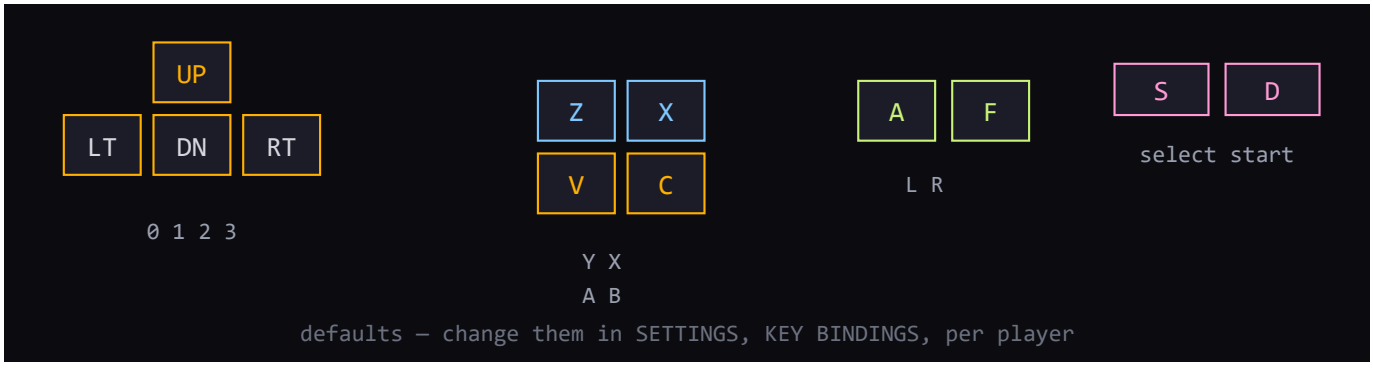
根列表：

页	它储存了什么
简介	保存和加载所有设置。新建、重命名、覆盖、删除。以文本形式复制或粘贴配置文件。
菜单	窗口、按键、菜单颜色、可选背景。

游戏面板	预览和播放窗口。
代码编辑器	重点提示。
绘画编辑器	画笔、调色板、地图视图。
特效编辑器	合成器和音符输入。
数据编辑器	车轮方向和颜色。
语言	菜单和提示语言。
许可证	许可文本，包括 Fusion Pixel 字体和 SDL3。

全球的：

- **布局**——面板排列，如上图所示。
- **音量**— 主音量，0 到 10。
- **按键绑定**——8名玩家的专属按键。按下按键即可锁定。方向键、ABXYLR、选择键。
- **呈现方式**——跳过空闲帧，或绘制每一帧。
- **全屏**——无边框。
- **刷新**——窗口帧率。左右步长分别为 60、90、120 和最大值。按回车键可输入 1000 以内的数字。0 表示无上限。
- **垂直同步**——等待帧与帧之间的显示。
- **缩放比例**——整数窗口乘数。
- **着色器**——编辑器中的 CRT 和滤镜，而不是游戏中的。
- **颜色**——菜单文本和高亮显示。
- **背景**——可选的PNG格式图片，尺寸为128像素，位于菜单后方。清除此图片可将其移除。
- **编辑器光标**——指针形状。
- **鼠标**——可以是浮动指针，也可以是像素对齐指针。
- **UI速度**——按住某个方向键在列表中重复播放的速度。
- **资源管理器**——文件列表从计算机根目录或 exe 文件所在的文件夹开始。



默认键盘布局。A键在菜单中是确认键。B键是返回键。在游戏中，A键对应键盘上的第4和第5键。选择键（S）是第10键，它不是菜单快捷键。

按钮	角色	默认键
0	左边	左箭头
1	正确的	右箭头
2	向上	向上箭头
3	向下	向下箭头
4	A，确认	V
5	B，返回	C
6	X	X
7	是	Z
8	拉	F
9	L	一个
10	选择	S
11	开始	D

建造

选择平台：Windows、Linux、Mac、Android、iOS 或 Web。

- **名称**— 构建文件的文件名。
- **作者**— 适用于 Windows 和 Linux 系统。这是保存文件的文件夹名称。
- **Steam 应用 ID** — 适用于 Windows 和 Linux 系统。空值表示未启用 Steamworks。已填写的 ID 会将 `achieve(0)`...`achieve(15)` 映射到 Steam 成就 ACH_0...ACH_15。
- **ID** — 平台要求提供捆绑包标识符时使用的标识符。
- **隐藏代码**——在桌面和移动设备的叠加层中。它会减慢游戏卡带的 `peek` 的运行速度。它并非锁定。
- **全屏模式**——导出的播放器将以全屏模式启动。
- **着色器**——最多三层，已烘焙到播放器中。
- **下载 .p37 文件**——仅限网页版。会在暂停菜单中显示下载选项。
- **图标**——来自精灵图的图块：图块、大小、x、y。或者您裁剪的 PNG 封面。
- **导出**——选择文件夹。菜单会显示"构建已保存"或"构建失败"。

构建游戏时请开启网络连接。启动画面可能会匿名向服务器查询此构建版本的哈希值是否已知。此检查有两个原因：一是保护我（PIX37 的作者）免受未经授权的复制品的侵害；二是保护游戏开发者的权益。在手机应用市场（例如 Play（Android）或 iOS）上，盗用他人游戏并冒充原创作品的情况越来越普遍。游戏仍可启动。如果设备离线，启动画面会提示游戏未连接。此提示也暗示玩家，该游戏可能并非来自合法渠道。这是一个简单的检查机制，以便在您的游戏被他人盗用并未经授权出售时，您可以在 Play（Android）、Steam 或 itch.io 等

应用商店提起诉讼。服务器会将哈希值的首次注册时间与哈希值本身一起存储，并且不会更改。较早注册的时间代表原始构建版本，之后注册的哈希值则代表后续复制版本。

我希望这项检查能够保护游戏开发者的利益。游戏引擎已经为他们做了很多工作，而这正是其中之一。我知道这可能会引起争议。如果社区认为移除这项微型DRM更好，我会照办。

之前编辑器中的旧游戏卡带在语言更改后可能无法启动。请在此编辑器中打开 `.pix37` 项目，并导出新的 `.p37` 项目。

连接到 PIX37.PARTY

连接到 PIX37.PARTY 的功能在应用内。浏览器中没有登录页面。您可以使用 Steam 会话券或 itch.io 账号登录，并使用鼠标和手柄进行操作。该服务会显示一个 2 到 16 个字符的唯一显示名称。许可证密钥和 Steam ID 不会对其他玩家显示。

服务条款结束后，中心区域包括收藏夹、目录、排名、玩家和您的个人资料。

开发者许可证对应的是游戏编辑者。你制作游戏，上传游戏，然后留下星星。玩家许可证对应的是玩家。爱心来自正在玩跑酷游戏的玩家。

- **目录**— 浏览游戏。筛选条件包括类型（最多三个）、屏幕尺寸、游戏卡带尺寸以及上传内容是否包含 `.pix37` 项目。
- **收藏夹**——你标记过的游戏。
- **上传**——需要开发者许可证。您需要发送 `.p37` 文件，并可附加 `.pix37` 项目文件。`.p37` 是压缩后的字节码，难以阅读，且可能被故意混淆。`.pix37` 是您实际使用的项目文件：包含源代码、图片和音频，以其他人可以打开和学习的形式呈现。发送此文件可以让其他创作者研究游戏，而不是压缩的游戏卡带。文件中包含标题、作者、类型、封面、图标和成就名称。上传的 RAM 数值是游戏卡带达到的峰值内存占用。
- **评论和成就**——在游戏页面上。登录后解锁的成就将同时保存在您的帐户和设备上。
- **排名**——开发者按星级或点赞数排序。每行同时显示两种计数。点击开发者姓名即可查看该开发者的游戏，按点赞数排序。
- **玩家**——按已解锁成就排序的账号。

代码

该语言是一种小型 Lua 风格的脚本语言。这部分内容从一个空项目开始，讲解语言规则，然后展示每个函数的简短示例。

第一个项目

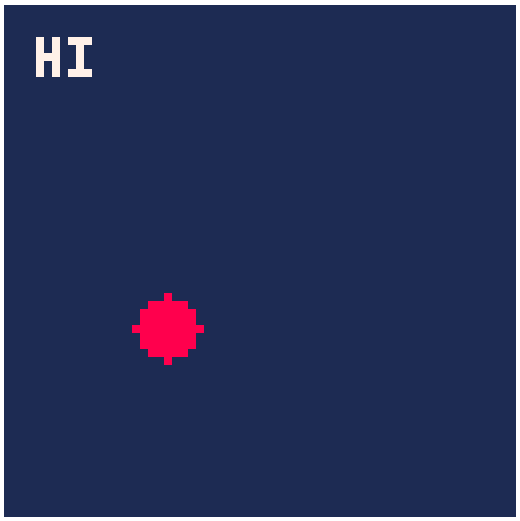
创建一个 64×64 像素、4 位颜色的项目。打开代码窗口。将标签页替换为：

```
_init()
  x = 20
  y = 40
end

_update()
  if btn(0) then x -= 1 end
  if btn(1) then x += 1 end
  if btn(2) then y -= 1 end
  if btn(3) then y += 1 end
end

_draw()
  cls(1)
  print("HI", 4, 4, 7)
  circfill(x, y, 4, 8)
end
```

按 Ctrl+R 运行程序。方向键移动圆圈。按 Esc 键停止程序并返回编辑器。



同样的三个调用：`cls(1)` 绘制深蓝色屏幕，`print` 绘制 HI，`circfill` 绘制粉红色圆圈。

什么节目，以及何时播出。

- `_init` 运行一次。
- `_update` 运行频率为 60 Hz。请在此处添加移动和按钮功能。
- `_draw` 用于绘制图像。如果没有指定帧率参数，则绘制速度取决于计算机和显示器的性能极限。`_draw(30)` 将绘制速度限制在 30 帧/秒，逻辑运算速度则保持在 60 帧/秒。
- `_set_fps(n)` 稍后会更改该上限。0 表示清除该上限。

名称不区分大小写；编辑器会以小写形式存储它们。`function` 和 `fun` 是同一个关键字。

语言

编程基础

程序是游戏卡带运行的一系列代码行的列表。每个标题首先显示代码行的形状，然后是一个简短的示例。以 `--` 开头的代码行是注释：给玩家的提示。游戏会跳过它。

变量

变量是一个用来存储值的名称。你可以自己选择变量名。后面的代码行可以读取这个变量名，也可以用同一个名称存储新的值。

该形状为 `name = value`。`local name = value` 将该名称保存在当前块内，因此游戏卡带的其他部分无法访问它。如果没有 `local`，则该名称是全局的，每个标签页都可以访问它。

```
hp = 3            -- global: the whole cart can read and change hp
local coins = 0   -- local: only this block can see coins
hp = hp + 1       -- read 3, add 1, store 4 back into hp
```

`nil` 表示没有值。`true` 和 `false` 分别表示“是”和“否”。从未设置过的名称会显示为 `nil`。

```
bonus = nil            -- nothing stored yet
if bonus == nil then   -- condition: is bonus empty?
  bonus = 0            -- body: store a number
end
```

表格

表格可以用同一个名称存储多个值。列表使用数字，第一个元素是 1，而不是 0。记录使用名称。一个表格可以包含另一个表格。

列表形状：`{ value, value }`。记录形状：`{ name = value, name = value }`。记录列表是一个表，其元素也是表。

```
-- a list. [1] is the first item
scores = {10, 20, 30}
print(scores[1], 1, 1, 7)      -- print draws text. this draws 10

-- a record. player.x is the field named x
player = {x = 10, y = 20}
print(player.x, 1, 8, 7)       -- draws 10

-- a list of records. enemies[1] is the first record
enemies = { {x = 4, y = 8}, {x = 20, y = 12} }
print(enemies[1].x, 1, 16, 7)  -- draws 4
print(enemies[2].y, 1, 24, 7)  -- draws 12
```

身体状况

条件是一个可以用"是"或"否"回答的问题，例如 `hp > 0` 或 `btnd(4)`。主体是指进入代码块时运行的一组代码行。`if` 和每个循环都使用相同的结构：一个条件，然后是一个主体，最后是 `end`。`repeat` 以 `until` 结束，而不是 `end`。

如果

`if` 仅在条件为真时执行其主体。`elseif` 尝试另一个条件。`else` 在所有条件均为假时执行。

形状：`if condition then`，然后是主体，然后是 `end`。

```
hp = 2
if hp > 0 then                 -- condition: is hp above zero?
  print("ok", 1, 1, 7)         -- body: print draws the word ok
elseif hp == 0 then            -- another condition
  print("empty", 1, 1, 7)
else
  print("no", 1, 1, 7)         -- body when every condition was false
end

if hp > 0 print(hp, 1, 32, 7)  -- one call, no then/end. draws 2
```

循环

循环会重复执行一个程序体。`while` 首先检查条件，如果条件为真则继续执行。`repeat` 先执行程序体，然后再进行检查，因此程序体至少执行一次。`for` 进行计数，最后一个数字会被计入。如果没有第三个数字，步长为 1，因此如果起始位置已经超过结束位置，则程序体不会执行。倒计时需要使用负步长。

```
n = 0
while n < 3 do         -- condition: keep going while n is under 3
  n += 1
  print(n)             -- 1, then 2, then 3: how many times the body has run
end

n = 0
repeat
  n += 1
  print(n)             -- body first, then the condition. prints 1, then 2, then 3
until n > 2

for i = 1, 3 do        -- i is 1, then 2, then 3
  print(i)
end

for i = 0, 4, 2 do     -- start, end, step of 2
  print(i)             -- 0, then 2, then 4
end

for i = 3, 1, -1 do    -- counting down needs the step
  print(i)             -- 3, then 2, then 1
end

for i = 1, 5 do
  if i == 3 then break end  -- leave the loop when i reaches 3
  print(i)                  -- 1, then 2
end
```

走过一张桌子

`all` 会遍历列表中的每个元素，从第一个开始。`pairs` 会遍历每个名称及其对应的值。

```
t = {1, 2, 3}                -- list: t[1] is 1, t[2] is 2, t[3] is 3
for v in all(t) do           -- v is 1, then 2, then 3
  print(v)                   -- print draws that number
end

player = {x = 10, y = 20}    -- record: a field x and a field y
for k, v in pairs(player) do
  print(k)                   -- k is the name, "x" then "y"
end
```

函数

函数是一个命名配方。其结构为 `function name(arguments)`，后跟一个函数体，再后跟 `end`。调用函数时，只需输入函数名和括号即可。`return` 会返回一个值，并且可以返回多个值。参数是该函数的局部变量。

```
function greet(who)       -- who is a local that holds what you pass in
  return "hi "..who       -- .. glues two texts. return sends the result back
end

print(greet("ada"))       -- draws hi ada
```

```
function pair()
  return 1, 2             -- two values
end

a, b = pair()             -- a becomes 1, b becomes 2
```

`local function` 是一个可以访问其周围局部变量的函数。这是一个闭包。一个表可以包含一个闭包，调用它时需要用点号 (.) 括起来。

```
function make()
  local n = 0
  local m = {}
  m.bump = function()     -- this function sees n
    n += 1
    return n
  end
  return m
end

counter = make()
print(counter.bump())     -- 1
print(counter.bump())     -- 2
```

是、否，以及选择

`not` 用于反转条件。`and` 仅当条件两边都为真时才为真。`or` 在条件任一边为真时为真。当 `a` 为真时，`a and b or c` 选择 `b`；否则选择 `c`。`cond ? a : b` 的原理相同，但简化为一行：当条件为真时选择 `a`，当条件不为真时选择 `b`。

```
hp = 2
hit = false
night = false

if hp > 0 and not hit then         -- both parts are true, so the body runs
  print("safe", 1, 1, 7)           -- draws safe
end

label = hp > 0 and "ok" or "out"   -- hp is above 0, so label is ok
print(label, 1, 8, 7)

shade = night ? 1 : 7              -- night is false, so shade is 7
print(shade, 1, 16, 7)
```

数学和文本

`+ - * /` 分别表示加、减、乘、除运算。`//` 表示除法运算，结果保留整数部分。`%` 表示余数。`^` 表示幂运算。`..` 用于连接两个文本。`#t` 用于统计列表中的项目数量。比较运算包括 `==`（相同）、`~=` 或 `!=`（不同）以及 `< > <= >=`。`x += 1` 将 x 加 1。同样的公式也适用于 `-= *= /= ^= %=`。

```
scores = {10, 20, 30}    -- the list named scores
tiles = 15 // 8          -- 1, the fraction is dropped
left = 15 % 8            -- 7, what is left over
title = "hi .."ada"      -- the text hi ada
n = #scores              -- how many items are in the list named scores
print(n, 1, 1, 7)        -- 3
print(title, 1, 8, 7)

score = 0
best = 10
if score ~= best then    -- condition: score is not the best
  score += 10            -- add 10 to score
end
print(score, 1, 16, 7)   -- 10
```

当地人居住的地方

`local` 会一直持续到 `end` 结束其代码块为止。`if` / `end` 则是一个独立的代码块。你可以在设置它的代码行上方引用一个局部变量，但在该代码行执行之前，它的值始终为 `nil`。

```
if true then
  print(coins)       -- nil: the next line has not run yet
  local coins = 5
  print(coins)       -- 5
end
-- coins does not exist out here
```

跳跃

`::label::` 标记一行。`goto label` 跳转到该行，但仅限于同一函数内部。

```
x = 0
::again::
x += 1                       -- add 1
if x < 3 then goto again end  -- condition: jump back while x is under 3
```

简而言之，就是记忆力。

`peek` 和 `poke` 通过一个 16 位地址读取实时像素。它们不保存游戏卡带的副本。地址 `0x5F00` 到 `0x5FFF` 是寄存器：相机寄存器、裁剪寄存器（只读）、绘制颜色寄存器、填充图案寄存器、三个调色板重映射寄存器以及页码寄存器。该地址两侧的 65280 个字节构成像素窗口。

当屏幕能够容纳在这 65280 字节内时（64×64 和 128×128 的屏幕可以，而 256×256 的屏幕则不行），屏幕从地址 0 开始逐行显示。剩余部分是 `0x5F41` 选择的精灵图（0 代表图 0）。如果该图大于剩余空间，则 `0x5F42` 会选择页面，此时地图不会显示在窗口中。如果图能够容纳，则尾部是 `0x5F43` 选择的地图的单元格，每个单元格占用一到两个字节；如果尾部空间不足，则 `0x5F44` 会将地图分页显示。

当屏幕内容无法完全显示时，整个窗口将显示为一个 65280 像素的页面。`poke(0x5F40, n)` 用于翻页。当 n 不为 0 时，`poke(0x5F41, n)` 会显示第 n 个工作表，而在此模式下，第 0 个工作表将不显示。当 n 不为 0 时，`poke(0x5F43, n)` 会显示地图 n，并且地图 n 会覆盖工作表。在此模式下，地图 0 仅显示为 `mget` 和 `mset`。

当 `peek` 未落在像素或寄存器上时，返回 0。`poke` 在此情况下不执行任何操作。精灵标志、音效和数据表不在此窗口中。请使用 `fget`、`sfxget` 和 `dget`。

实际用途

本页底部列出了所有函数，并附有PIX37函数的实际应用示例。每个示例下方的链接可查看更多关于该函数的信息。



流畅的动作

俯视图，128×128像素的游戏卡带。第一个精灵已经绘制完成。箭头可以向任意方向加速，摩擦力会减慢速度。精灵会环绕显示：底部边缘的一半会绘制在顶部，其他边缘也一样。

```
_init _update _draw btn mid cls spr palt
```



检查下一步是否被阻挡。

一张 64×64 的游戏卡带。地板和右侧的墙壁已经绘制在地图上，使用精灵 3 绘制，并且该精灵上设置了标志 0。箭头控制精灵 1 移动。顶部的文字显示，如果下一步会与实心图块重叠，则显示“阻塞”；如果不会重叠，则显示“自由”。

```
flr fget mget btn cls map spr print _update _draw
```



用翻转擦拭

按下 A 键。擦除效果由 flip() 函数绘制，一次绘制一行黑色线条，而不是在 _draw() 函数内部绘制。_draw() 函数在 flip() 函数运行时不会干扰擦除效果，因此每一行都会始终显示在屏幕上。

```
rectfill flip btnd cls print _update _draw
```



函数表（类）

精灵 1 已经绘制完成。`make_monster` 函数的作用类似于一个类。它返回一个表格，`update` 和 `draw` 方法用于维护该怪物的 x 和 y 坐标。表格顶部的数字表示它已经走了多远。

```
spr cls print flr _init _update _draw
```



解读按钮的四种方法

A键是按钮4。按住它，充能增加。松开它，充能归零。按住A键，上下移动会不断重复，菜单行也会持续移动。btnd是按键的第一帧，在这个游戏卡带中，它仅用作按键的起始帧。

```
btn btnd btnp btnu cls print _update _draw
```



用鼠标射击

精灵 1 已绘制在中间。一个白点是光标。单击鼠标左键会向光标发射一个像素。角落里显示的滚轮可以改变速度。

```
scroll clickl mouse sqrt add all cls spr pset print _update _draw
```



输入名称并确认

按 A 键打开输入框。上下键切换闪烁的字母。A 键输入，X 键删除，Y 键空格，B 键确认。然后，输入框会显示你的名字。

btnd keyon keystr flr time cls print _update _draw



镜头对准玩家

一个 128×128 像素的游戏卡带。精灵 1 初始位于瓷砖地板的中央。箭头会移动。摄像机始终保持精灵居中。角落的 x 和 y 坐标不会滚动，因为不带参数的 camera() 函数会将视图切换回 HUD 显示区域。

btn screen camera cls map spr print flr _update _draw



视觉小说中的几行文字

文本以数据格式呈现，语言为英语和波兰语。A 键表示继续，或在屏幕上出现选项时选择“保持”。B 键表示在出现选项时退出，否则切换语言。X 键表示在显示选项时切换语言。对话框使用 8px 字体，因此波兰语字母会显示出来。

dstr dget btnd cls print8 print _update _draw



用戳戳法作画

这是一个 64×64 的游戏卡带，地址 0 是左上角的像素，下一行从 64 开始。游戏卡带写入像素，复制一行，清除一行，并打印它读取的值。

cls poke peek poke2 peek2 poke4 peek4 memcpy memset print _draw



您可以添加到列表中

A 负责添加一个数字，最大不超过 8。B 负责删除最后一个数字。X 负责删除第一个数字。列表会显示在屏幕上。

btnd count add deli cls ipairs print pack unpack next foreach select pcall assert del _update _draw



一个被拆解的名字

游戏卡带将 ada、12、blue 分开，并打印出名称、其首字母、12 和单词 blue。

cls split sub chr ord tonum tostr print _draw



弹奏一个音符，然后改变它

模式 0 为一个音符，每次重复播放的音量都相同。A 在该音符结束后再次演奏。B 提高音调并演奏该音符。

sfxset sfxhdr sfx btnd sfxstat sfxget cls print _init _update _draw



一首写在游戏卡带里的歌

第 0 行循环播放两个乐句，分别对应前两个通道。行号保持在 0。ch0 和 ch1 分别代表每个通道当前所在的音符。在游戏卡带中编辑乐句、LEN 和 LOOP 参数，可以查看这些参数在音乐面板中的实际作用。

`songget songset music musicstat cls print _init _draw`



数据中的三个怪物

数据面板中已有名称列和生命值列，分别包含史莱姆、蝙蝠和眼睛。A 键会伤害史莱姆。右侧的数字是它的生命值。

`drows dstr dget add btnd dset dcols cls print all _init _update _draw`



记住比分

A 加了一分并保存。退出游戏后再次打开游戏卡带：分数还在。B 忘记了这件事。

`load btnd save delsave cls print _init _update _draw`




另一个文件中的第二章

两个游戏卡带共用一个存档位。第一部分：A 表示游戏完成。第二部分（先打开）提示完成第一部分。之后，第二部分打开，B 会清除存档，以便重新开始第一部分。

`load saveslot btnd save delsave cls print _init _update _draw`



一项成就

第 8 个精灵已经绘制完成。按 A 键十次。屏幕上会显示成就名称，并且由于第 8 个精灵已解锁，因此它会保留下来。按 B 键会清除本地文件中的该成就并重置计数。

`achdef btnd achieved achieve achclear cls print spr _init _update _draw`



你可以走进去的房间

256×256像素的游戏卡带。白色圆点是光标。菜单选项包括“创建房间”、“加入房间”和“服务器列表”。“创建房间”会打开一个名为“walk”的房间并开始游戏。“加入房间”会滚动显示一个地址（数字、一个点、一个冒号），按下“B”键连接。服务器列表上下移动，“A”键加入当前行。在游戏中，箭头指向第一个角色“walk”。游戏共享一枚硬币：由创建房间的玩家移动，拾取硬币会使该玩家的分数加1，并生成下一个角色。每个角色（包括你自己的角色）旁边都会显示分数。角落会显示当前玩家人数、局域网地址和创建房间的公网地址。按下“B”键离开：你和其他玩家返回菜单，下一个创建房间的玩家将创建一个新的房间，初始分数为0。

`btn btnd mouse pset spr print flr rnd pairs nlisten nhost nannounce nlist njoin nleave nsend nkb nkbstr whoami nplayers nlan _mp _mp_join _mp_leave _mp_fail palt cls time _init _update _draw`



等待的对话

A 键显示下一行。函数会在每次执行 yield 操作后暂停，直到您再次按下 A 键。

```
yield cocreate btnd coresume cls print _init _update _draw
```



稍等半秒钟

大约半秒钟后，屏幕上的文字从“...”变成了“hi”。协程暂停时，屏幕仍在继续绘制。

```
yield cocreate costatus coresume cls print _init _update _draw
```



一个个出现的圆圈

五个圆圈从顶部滴落，彼此间隔两帧，然后落下。

```
add yield cocreate coresume all cls circfill _init _update _draw
```



时钟上的两帧画面

精灵1和精灵2是同一个角色的两帧图像。图像每秒切换四次。

```
cls flr time spr _draw
```

每个功能

方括号内的参数是可选的。表格和字符串的索引从 1 开始，但 DATA 类型的行和列的索引从 0 开始。颜色是调色板索引。

指数

_draw	全部	循环填充
_init	断言	点击1
_mp	阿坦2	点击
_mp_失败	乐队	点击器
_mp_加入	不	剪辑
_mp_离开	博尔	CLS
_mp_更新	按钮	共同创造
_设置_fps	宾德	颜色
_更新	BTNP	核心简历
腹肌	BTNU	因斯
阿赫清除	异或	肋骨
阿赫德夫	相机	计数
达到	天花板	光标
达到了	chr	dcols
添加	循环	德尔

熟食店
德尔保存
dget
卓尔精灵
数据集
数据线
得到
填充
翻转
佛罗里达州
foreach
偏移量
信息
ipairs
断键
凯永
按键状态
按键字符串
线
负载
日志
LSHR
地图
最大
内存复制
内存设置
管理
中
分钟
鼠标
设定值
音乐
音乐统计
通知
呼叫
下一个
主机
联结
恩克布
恩克伯斯特
恩兰
恩利夫
列表
听
注释
玩家
恩森德
恩斯顿
秩序
椭圆形
椭圆形填充
包
对
朋友
帕尔格布
帕尔特
呼叫
偷看
偷看2
窥视4
性能
普吉特
戳
戳2
戳4
打印
打印10
打印8
普塞特
直角
直填充
rnd
罗特尔
罗特尔
正确
正确填充
保存
保存槽
屏幕
滚动
选择
设置图
设置调色板
设定值
塞特克斯
转向
音效
sfxget
sfxhdr
sfxset
sfxstat
sget
信号
谢尔
显示统计数据
什尔
罪恶
松格特
歌曲集
分裂
斯普尔
开方
斯兰德
资产
sspr
统计数据
子
时间
线
托努姆
托斯特尔
打开包装
瓦阿米
产量

生命周期

_init

_init() ... end

游戏卡带启动时运行一次。位于任何钩子函数之外的代码也会被合并到 _init 函数中。

```
_init()
  x = 64
  y = 40
end
```

_update

_update() ... end

游戏逻辑。即使 _draw 被限制或主机速度较慢，也会以 60 Hz 的频率调用。在此处读取按钮。

```
_update()
  if btn(0) then x -= 1 end
  if btn(1) then x += 1 end
end
```

_draw

_draw([fps]) ... end

绘制一帧。如果不带参数或参数为 0，则以主机提供的频率运行。传递参数 n（1 到 999）可将绘制频率限制为每秒 n 帧。_update 函数的刷新率保持在 60 Hz。

```
_draw(30)
  cls(1)
  spr(1, x, y)
end
```

_set_fps

_set_fps(n)

游戏运行时更改_draw上限。0表示不设上限。n值限制在0到999之间。

```
if btn(4) then _set_fps(60) end
```

flip

```
flip()
```

呈现屏幕并等待下一帧。仅当您绘制自己的帧循环时才使用此方法。之后不要进行忙等待。

```
cls(0)
print("tick", 4, 4, 7)
flip()
```

time

```
time()
```

自 _init 以来的秒数。

```
print(time(), 4, 4, 7)
```

输入

btn

```
btn([b], [p])
```

按住按钮 b 时为 1，否则为 0。p 是玩家，0..7。不带参数时，它返回玩家 0 的位掩码。按钮：0 左，1 右，2 上，3 下，4 A，5 B，6 X，7 Y，8 R，9 L，10 选择，11 开始。

```
if btn(4) then shoot() end
```

btnd

```
btnd(b, [p])
```

仅在按钮 b 按下的 _update 帧中为真。

```
if btnd(5) then menu_open = not menu_open end
```

btnp

`btnp([b], [p])`

按下按钮时返回真，按住按钮时再次返回真：延迟 30 个时钟周期，然后每 8 个时钟周期返回一次。不按下按钮时返回一个掩码。

```
if btnp(2) then cursor -= 1 end
```

btnu

`btnu(b, [p])`

仅在按钮松开的瞬间有效。

```
if btnu(4) then charge = 0 end
```

mouse

`mouse()`

返回鼠标在游戏像素中的 x、y 坐标。

```
mx, my = mouse()
pset(mx, my, 7)
```

clickl

`clickl([p])`

鼠标左键。clickl() 被按住。clickl(1) 是按下的边缘。

```
if clickl(1) then place(mouse()) end
```

clickr

```
clickr([p])
```

鼠标右键。与 clickl 相同的 1 参数边缘。

```
if clickr() then cancel() end
```

clickm

```
clickm([p])
```

按下中键，或单击滚轮。

```
if clickm(1) then drag = true end
```

scroll

```
scroll()
```

本帧中鼠标滚轮的移动。正负取决于滚轮的滚动方向。

```
cam_y -= scroll()
```

keyon

```
keyon()
```

打开一个由按钮控制的文本框。上下键切换字母 A-Z。A 键输入该字母。X 键删除。Y 键空格。B 键确认并关闭文本框。键盘无法在此文本框中输入内容。

```
if btnd(4) then keyon() end
```

keyoff

`keyoff()`

隐藏文本框。

```
keyoff()
```

keystr

`keystr([s])`

字段中输入文本，然后帧 B 上的 1 表示确认，之后上下字母会发生变化。字段关闭后，该字母为空。keystr("") 清除文本。

```
name, done, letter = keystr()
if done == 1 then start(name) end
```

keystat

`keystat()`

文本字段打开时为 1，否则为 0。

```
if keystat() == 1 then return end
```

绘画

screen

`screen()`

当前游戏卡带屏幕尺寸（以像素为单位）。返回宽度，然后返回高度。这是游戏分辨率，而非编辑器窗口分辨率。

```
w, h = screen()
```

cls

`cls([col])`

清屏。默认颜色为0。

```
cls(1)
```

color

`color(col)`

设置后续调用省略 col 参数时使用的绘制颜色。不带参数时，返回当前颜色。

```
color(7)
print("hi")
```

camera

`camera([x, y])`

每次绘制调用都会进行偏移。camera() 重置为 0, 0。

```
camera(cam_x, cam_y)
```

clip

`clip([x, y, w, h, clip])`

将绘制区域限制在矩形内。第五个非零参数表示与当前裁剪区域相交。`clip()` 函数恢复全屏显示。裁剪区域寄存器为只读。

```
clip(8, 8, 48, 48)
circfill(32, 32, 20, 8)
clip()
```

cursor

```
cursor([x, y, col])
```

设置下一个不带坐标的打印位置，还可以选择设置颜色。cursor() 返回 x、y 坐标。

```
cursor(4, 4, 7)
print("line 1")
print("line 2")
```

pset

```
pset(x, y, [col])
```

设置一个屏幕像素。计入像素统计数据。

```
pset(x, y, 10)
```

pget

```
pget(x, y)
```

屏幕像素在 x、y 处的颜色。

```
if pget(x, y+8) == 3 then grounded = true end
```

print

```
print(v, [x, y, col])
```

绘制 3×5 像素的文本（UTF-8 编码，取自 Fusion 字体）。如果不设置 x、y 坐标，则会在光标处打印文本并换行。

```
print("score "..score, 2, 2, 7)
```

log

`log(...)`

将参数（以空格分隔）追加到文本缓冲区。pix37_run 会将该缓冲区的内容打印到标准输出。它不会进行任何绘制操作。

```
log("hp", hp)
```

print8

`print8(v, [x, y, col])`

与印刷版相同，使用 8 像素的 Fusion 字体。

```
print8("pause", 16, 28, 7)
```

print10

`print10(v, [x, y, col])`

与印刷版相同，使用 10 像素的 Fusion 字体。

```
print10("pix37", 8, 8, 10)
```

line

`line([x0, y0, x1, y1, col])`

从 x0,y0 到 x1,y1 的直线。两个参数从上一端继续传递。line() 函数会重置该传递链。

```
line(0, 32, 63, 32, 6)
```

circ

`circ(x, y, r, [col])`

圆形轮廓。半径单位为像素。

```
circ(32, 32, 12, 12)
```

circfill

`circfill(x, y, r, [col])`

实心圆。

```
circfill(x, y, 3, 8)
```

oval

`oval(x0, y0, x1, y1, [col])`

在边界框内绘制椭圆轮廓。包含四个角。

```
oval(8, 20, 56, 44, 7)
```

ovalfill

`ovalfill(x0, y0, x1, y1, [col])`

边界框内的填充椭圆。

```
ovalfill(10, 40, 54, 60, 3)
```

rect

`rect(x0, y0, x1, y1, [col])`

矩形轮廓。四个角包含在内。

```
rect(0, 0, 63, 63, 7)
```

rectfill

```
rectfill(x0, y0, x1, y1, [col])
```

填充矩形。

```
rectfill(0, 56, 63, 63, 3)
```

rrect

```
rrect(x, y, w, h, [r, col])
```

圆角矩形轮廓。x、y 坐标为左上角坐标。w 和 h 为尺寸（单位：像素），两者均必须大于 0。r 为圆角半径，默认值为 0，半径限制为短边的一半。

```
rrect(8, 8, 48, 16, 3, 7)
```

rrectfill

```
rrectfill(x, y, w, h, [r, col])
```

填充圆角矩形。参数与 rect 相同。

```
rrectfill(8, 40, 48, 16, 2, 1)
```

fillp

```
fillp([p])
```

用于后续填充形状的 16 位填充图案。0 表示实心。fillp() 返回当前图案。位遍历 4×4 图案。

```
fillp(0xA5A5)
rectfill(8, 8, 40, 40, 10)
fillp(0)
```

pal

`pal([c0, c1, [mode]])`

将颜色 c0 重新映射到 c1。模式 0 是绘制调色板，1 是显示调色板，2 是填充图案调色板。pal() 重置。

```
pal(8, 10)
circfill(32, 32, 8)
```

palrgb

`palrgb(col, r, g, b)`

将一个调色板条目设置为 RGB，每个通道为 0..255。

```
palrgb(8, 255, 80, 40)
```

palt

`palt([col, t])`

透明度。`palt(col)` 或 `palt(col, true)` 使颜色 `col` 透明。`palt(col, false)` 使其不透明。`palt()` 使所有颜色都不透明。

```
palt(0)
spr(1, x, y)
```

setpalette

`setpalette(n)`

选择调色板库 n。setpalette() 返回当前库。

```
setpalette(1)
```

tline

`tline(x0, y0, x1, y1, mx, my, [mdx, mdy])`

在屏幕上绘制一条线，同时从地图位置 mx、my、mdx、mdy 开始采样地图像素，并沿着该线进行地图采样。

```
tline(0, 40, 63, 20, 0, 0, 0.5, 0)
```

精灵和地图

spr

`spr(n, x, y, [w, h, fx, fy, tex, rot])`

在 x、y 坐标处绘制精灵 n。w 和 h 是像素大小，默认值为 8。fx 和 fy 非零时分别进行水平和垂直翻转。tex 是精灵图的标签页。rot 是旋转角度（0 到 1），方向与 sin 和 cos 相同。0 表示精灵不旋转。

```
spr(1, x, y, 8, 8, 0, 0, 0, aim)
```

sspr

`sspr(sx, sy, sw, sh, dx, dy, [dw, dh, fx, fy, tex])`

将精灵图中的矩形复制到屏幕上，缩放比例为 dw 乘以 dh。默认情况下保持源尺寸不变。

```
sspr(0, 0, 8, 8, 16, 16, 32, 32)
```

sget

`sget(x, y, [tex])`

精灵图上一个纹素的颜色。

```
c = sget(0, 0)
```

sset

```
sset(x, y, [col, tex])
```

在精灵图上写入一个纹素。

```
sset(3, 3, 8)
```

fget

```
fget(n, [f, tex])
```

精灵标志。fget(n) 返回标志字节。当标志位 f（0..7）被设置时，fget(n, f) 为真。

```
if fget(tile, 0) then solid = true end
```

fset

```
fset(n, [f], v, [tex])
```

设置精灵标志。fset(n, v) 写入整个字节。fset(n, f, v) 设置一位。

```
fset(2, 0, true)
```

setspr

```
setspr(n)
```

选择当前激活的精灵图。setspr() 返回当前精灵图。与 settex 使用相同的库。

```
setspr(1)
```

settex

```
settex(n)
```

setspr 的别名。

```
settex(0)
```

map

```
map(cx, cy, sx, sy, [cw, ch, layer, tex])
```

绘制地图。地图单元格 cx、cy 绘制在屏幕 sx、sy 上。cw、ch 为 0 表示绘制整个地图。图层被接受或忽略。tex 是图块集。

```
map(0, 0, 0, 0, 16, 16)
```

mget

```
mget(cx, cy)
```

地图单元格 cx、cy 中的精灵 ID。

```
if fget(mget(tx, ty), 0) then bounce() end
```

mset

```
mset(cx, cy, n)
```

将精灵 n 写入地图单元格 cx, cy。

```
mset(3, 4, 0)
```

setmap

```
setmap(n)
```

选择地图 n。setmap() 返回当前地图。地图 0 是一个实际存在的地图。

```
setmap(1)
```

记忆

peek

`peek(addr)`

读取 64 KB 窗口的一个字节。窗口首先是实时屏幕，然后是精灵像素，最后是映射单元格（当屏幕大小为 65280 字节时）。更大的屏幕使用 poke(0x5F40, page) 进行分页。未映射的地址返回 0。也写作 @addr。

```
px = peek(0)
```

poke

`poke(addr, ...)`

在地址 addr 处写入字节。额外参数会写入后续字节。未映射的地址将被忽略。同时写入 $addr 和 val。点击屏幕也算作一次像素写入。

```
poke(0x5F00, 4) -- camera x
```

peek2

`peek2(addr)`

无符号 16 位值，小端序，地址为 addr。

```
w = peek2(0x6000)
```

poke2

`poke2(addr, val)`

写入一个无符号 16 位值，小端序。

```
poke2(0x6000, 1000)
```

peek4

`peek4(addr)`

无符号 32 位值，小端序。

```
n = peek4(0x6000)
```

poke4

`poke4(addr, val)`

写入一个无符号 32 位值，小端序。

```
poke4(0x6000, 1)
```

memcpy

`memcpy(dst, src, len)`

在同一个 64 KB 窗口内复制 len 字节。

```
memcpy(0x6000, 0x6100, 64)
```

memset

`memset(dest, val, len)`

从 dest 开始，用 val 填充 len 个字节。

```
memset(0, 0, 64)
```

数学

abs

`abs(x)`

绝对值。

```
d = abs(tx - x)
```

flr

`flr(x)`

向负无穷大方向取整。

```
tx = flr(x / 8)
```

ceil

`ceil(x)`

向正无穷方向取整。

```
n = ceil(hp / 8)
```

max

`max(a, b)`

取两者中较大的值。

```
x = max(0, x)
```

min

`min(a, b)`

取两者中较小的值。

```
x = min(63, x)
```

mid

`mid(a, b, c)`

三个数值中的中间值。可用作夹具。

```
x = mid(0, x, 63)
```

sgn

`sgn(x)`

x 的符号：-1、0 或 1。

```
x += sgn(dir)
```

sqrt

`sqrt(x)`

平方根。负数参数不是一个数。

```
dist = sqrt(dx*dx + dy*dy)
```

sin

`sin(x)`

正弦。x 的取值分别为：0、0.25、0.5、0.75，绕圆一周。不是弧度。

```
y = 32 + sin(time()) * 8
```

cos

`cos(x)`

余弦函数，反过来，与正弦函数遵循相同的约定。

```
x = 32 + cos(time()) * 8
```

atan2

```
atan2(dx, dy)
```

向量 dx、dy 的角度，从 0 到 1 依次变化。

```
aim = atan2(mx - x, my - y)
```

rnd

```
rnd([x])
```

随机数，取值范围为 [0, x)。默认值 x 为 1，因此 rnd() 的取值范围为 0..1。对于骰子，flr(rnd(6))+1。

```
c = flr(rnd(16))
```

srand

```
srand(seed)
```

设置随机数生成器的种子，以便后续的 rnd 序列可以重复执行。

```
srand(1)
```

band

```
band(a, b)
```

按位与运算。也写作 a & b。

```
if band(flags, 1) ~= 0 then hit() end
```

bor

```
bor(a, b)
```

按位或运算。也写作 a | b。

```
flags = bor(flags, 4)
```

bxor

`bxor(a, b)`

按位异或。也写作 a ^^ b。

```
n = bxor(n, 255)
```

bnot

`bnot(a)`

按位非。也写作 ~a。

```
mask = bnot(1)
```

shl

`shl(a, n)`

向左移动。也可以写成 << n。

shr

`shr(a, n)`

算术右移。也写作 >> n。

```
n = shr(n, 1)
```

lshr

`lshr(a, n)`

逻辑右移。也写成 >>> n。

```
n = lshr(n, 4)
```

rotl

```
rotl(a, n)
```

将位向左旋转。也可以写成 <<> n。

```
n = rotl(n, 1)
```

rotr

```
rotr(a, n)
```

将位向右旋转。也可以写成 >>< n。

表格和值

add

```
add(t, v, [i])
```

将 v 追加到表 t 中。如果给定 i，则插入到从 1 开始的索引处。

```
add(bullets, {x=x, y=y})
```

del

```
del(t, v)
```

删除第一个等于 v 的条目。

```
del(items, gem)
```

deli

```
deli(t, [i])
```

移除索引为 i 的条目并返回该值。如果未指定索引 i，则移除最后一个条目。

```
local b = deli(bullets, i)
```

all

```
all(t)
```

按顺序遍历数组部分。遇到空位时，遍历会停止，就像从 1 开始计数的数组结束时那样。

```
for b in all(bullets) do
  b.x += 2
end
```

foreach

```
foreach(t, fn)
```

对数组中的每个元素调用 fn(v)。

```
foreach(enemies, draw_enemy)
```

pairs

```
pairs(t)
```

遍历每个键和值，包括命名字段。

```
for k, v in pairs(player) do
  print(k)
end
```

ipairs

```
ipairs(t)
```

使用从 1 开始的索引遍历数组部分。

```
for i, v in ipairs(deck) do
  spr(v, i*8, 8)
end
```

next

`next(t, [k])`

返回第 k 个键值对之后的下一个键和值。`next(t)` 从开头开始。用于键值对。

```
k, v = next(t)
```

count

`count(t, [v])`

数组部分的长度。如果给定变量 v，则统计有多少个元素的值等于 v。

```
if count(gems) == 0 then win() end
```

pack

`pack(...)`

将参数打包到一个新表中。

```
t = pack(1, 2, 3)
```

unpack

`unpack(t, [i, j])`

返回 t[i] 到 t[j] 的各个值。默认值涵盖整个数组部分。

```
a, b = unpack(t)
```

select

`select(i, ...)`

select("#", ...) 返回后面有多少个参数。select(i, ...) 返回从 i 开始的参数。

```
n = select("#", 1, 2, 3)
```

assert

```
assert(c, [msg])
```

当 c 为 false 或 nil 时，程序停止并报错。msg 替换默认文本。

```
assert(hp > 0, "dead")
```

pcall

```
pcall(f, ...)
```

调用 f。成功时返回 true 和结果。出错时返回 false 和错误文本。

```
ok, err = pcall(load_level, 3)
```

字符串

chr

```
chr(n)
```

字节码 0..255 中的一个字符。

```
s = chr(65) -- A
```

ord

```
ord(s, [i, j])
```

从索引 i 到 j 的 s 的字节码。索引从 1 开始。默认值为第一个字符。

```
code = ord("a")
```

sub

`sub(s, i, [j])`

从 i 到 j 截取子字符串，从 1 开始。省略 j 则截取尾部。

```
name = sub(keystr(), 1, 8)
```

split

`split(s, [sep, conv])`

将数据分割成表格。分隔符默认为逗号。分隔符可以是数字，表示数据块的宽度。如果 `conv` 为真，则使用 `tonum` 转换数据块。

```
words = split("red,blue", ",")
```

tonum

`tonum(s, [base])`

解析一个数字。基数可以是 2 到 36。如果文本不是数字，则返回 nil。

```
n = tonum("255")
```

tostr

`tostr(v, [digits])`

将数值转换为文本。digits 设置数字的小数位数。

```
print(tostr(hp, 0), 4, 4, 7)
```

声音的

sfx

```
sfx(n, [ch, offset, len])
```

播放模式 n 并返回通道。省略 ch 或传递 -1，则选择第一个空闲通道。ch -2 会停止所有通道上的模式 n。n -1 会停止一个通道，如果省略 ch，则会停止所有通道。n -2 会让循环播放的模式结束，而不是继续循环。不带参数的 sfx() 返回 -1。

```
ch = sfx(0)
```

sfxget

```
sfxget(n, step)
```

返回一个包含一个模式步骤的表格：[1] 音高，[2] 波形，[3] 音量，[4] 效果。音高范围为 0..63（C-0 到 D#5）。波形范围为 0..15。音量范围为 0..7，0 表示静音。效果范围为 0..7。

```
step = sfxget(0, 0)
pitch = step[1]
```

sfxset

```
sfxset(n, step, pitch, wave, vol, fx)
```

编写一个模式步骤。所有六个参数均为必需参数。

```
sfxset(0, 0, 24, 3, 5, 0)
```

sfxhdr

```
sfxhdr(n, [key, val])
```

模式头。sfxhdr(n) 返回头表。sfxhdr(n, key, val) 设置一个字段：speed、loop_in、loop_out、level、noise、crush、dtune、rverb、dampn、swing、steps。数字 0 到 10 按顺序选择相同的字段。speed 的单位是 TIC，即每步的节拍数。数值越高，速度越慢。15 表示每分钟 120 拍。16 位模式接受 1 到 255 的值。较窄的合成器会根据其 TIC 列表进行调整。

```
sfxhdr(0, "speed", 15)
```

sfxstat

```
sfxstat([ch])
```

模式 ID 在频道 ch 上播放，或 -1。默认频道为 0。

```
if sfxstat(0) < 0 then sfx(1) end
```

note

```
note(pitch, [wave, vol, fx, ch, ticks])
```

播放一个无节奏型的单音。音高 0..63，波形 0..15，音量 0..7，效果 0..7。“ticks”表示持续时间，与 TIC 的 120 Hz 节拍相同。省略“ticks”参数，则持续时间为 15，即一个默认步长。效果会持续这段时间，然后通道静音。

```
note(36, 3, 5, 5, 0)
```

music

```
music([n, fade, mask])
```

从头开始播放歌曲 n。`music(-1)` 停止播放。如果调用 `music` 并传入当前正在播放歌曲的 ID，则歌曲会继续播放。`music()` 返回当前正在播放的歌曲 ID，如果返回则返回 -1。`fade` 参数为毫秒：歌曲淡入，当 `n` 为 -1 时淡出。`mask` 位表示歌曲使用的声道。0 表示使用全部八个声道。其他声道则用于音效。

```
music(0, 1000, 7)
```

musicstat

```
musicstat()
```

正在播放的歌曲表。song 和 row 分别代表歌曲 ID 和行号，如果当前没有歌曲播放，则为 -1。step 和 pos 均为八位数字。step[1] 代表通道 0，step[8] 代表通道 7。step 代表音符索引，如果该通道静音，则为 -1。pos 代表该音符的播放进度，范围从 0 到 1。tick 代表当前行已播放的节拍数。played 代表自 music() 函数调用以来已播放的行数，包括当前播放的行。

```
st = musicstat()
row = st.row
note = st.step[1]
```

songget

```
songget(song, [row, ch])
```

歌曲单元格中存储了模式 ID。ch 的取值范围为 0 到 7。255 表示单元格为空。这是实时调用；代码提示文本仍然显示的是旧的参数顺序。

```
id = songget(0, 0, 0)
```

songset

```
songset(song, row, ch, pattern)
```

将模式 ID 写入歌曲单元格。使用 255 清除单元格。

```
songset(0, 0, 0, 1)
```

stat

```
stat(n)
```

仅音频。16..23 是通道 n-16 的音效 ID。24..31 是通道 n-24 的音乐模式。任何其他 n 值均返回 0。帧定时和 RAM 是性能和信息，而非统计数据。

```
id = stat(16)
```

数据表

dget

```
dget(t, row, col)
```

从 DATA 面板读取数据表 t 的一个单元格。行和列从 0 开始。

```
hp = dget(0, 0, 1)
```

dset

```
dset(t, row, col, v)
```

写入单元格。文本列可以使用字符串。

```
dset(0, 0, 0, "slime")
```

dstr

```
dstr(t, row, col)
```

单元格内容为文本。数字会被转换。

```
name = dstr(0, row, 0)
```

drows

```
drows(t)
```

表t有多少行?

```
for r = 0, drows(0) - 1 do
end
```

dcols

```
dcols(t)
```

表t有多少列?

```
cols = dcols(0)
```

保存和成就

save

`save(k, v)`

将值 v 存储在保存文件的键 k 下。v 可以是数字、字符串或数字与字符串的表格。文件已压缩。

```
save("hp", hp)
```

load

`load(k)`

读取键 k。如果键不存在，则返回 nil。

```
hp = load("hp") or 3
```

delsave

`delsave(k)`

从存档中删除键 k。

```
delsave("hp")
```

saveslot

`saveslot([name])`

选择作者文件夹内的存档文件。如果未指定文件名，则默认为游戏标题。如果该文件已存在，则返回 1。

```
if saveslot("slot1") == 1 then
  hp = load("hp") or 3
end
```

achieve

```
achieve(i)
```

解锁成就 i，数值范围为 0 到 15。存储在本地。Steam 版本也会设置 ACH_i。目录会话会将此信息报告给玩家帐户。

```
if coins >= 100 then achieve(0) end
```

achieved

```
achieved(i)
```

1 当成就 i 已解锁时。

```
if achieved(0) == 1 then spr(8, 4, 4) end
```

achdef

```
achdef(i, text, [spr, tex])
```

成就 i 的提示文字，从屏幕底部显示 15 秒。spr 是一个可选的 8×8 图块，tex 是它的图块。

```
achdef(0, "rich", 8)
```

achclear

```
achclear(i)
```

清除本地文件中的成就 i。已设置的 Steam ACH_i 值保持不变。

```
if btnd(5) then achclear(0) end
```

网络

whoami

whoami()

在线会话期间，您的玩家 ID 为 0 到 15。离线时，玩家 ID 为 -1。

```
id = whoami()
```

nhost

nhost()

如果这台机器是主机，则为 1，否则为 0。

```
if nhost() == 1 then spawn() end
```

nplayers

nplayers()

0..15 范围内有多少个活跃 ID?

```
print(nplayers(), 4, 4, 7)
```

nlisten

nlisten()

开始托管 UDP 并返回 LAN 地址（ip:端口）。

```
addr = nlisten()
```

njoin

njoin(addr)

加入 IP 地址或 IP:端口。

```
njoin("10.0.0.8:7777")
```

nleave

```
nleave()
```

离开会话。其他用户看到 _mp_leave，原因为 0。

```
if btnd(11) then nleave() end
```

nsend

```
nsend([p,] name, ...)
```

发送有损消息。名称为事件。最多可添加四个附加值。p 值介于 0 到 15 之间，表示向目标节点发送一条消息；省略 p 值则发送给所有人。

```
nsend("pos", x, y)
```

ncall

```
ncall([p,] name, ...)
```

与 nsend 的格式相同，但消息会不断重试直到收到确认。连续三次失败会导致会话终止。

```
ncall("hit", target, dmg)
```

nlan

```
nlan()
```

nlisten 后的当前 LAN IP:端口。

```
print(nlan(), 2, 2, 7)
```

nstun

`nstun()`

您的公网 IP 地址和端口号来自 STUN。其他网络上的访客可以尝试加入该网络。

```
pub = nstun()
```

setturn

`setturn(addr, [user, pass])`

当直接路径失败时，TURN 会作为备用路径。使用空参数调用 `setturn()` 函数会关闭此功能。地址格式为主机:端口。

```
setturn("turn.example:3478", user, pass)
```

nlist

`nlist(game)`

该游戏名称的开放会话列表。每个条目包含地址、玩家和名称。

```
rooms = nlist("duel")
```

nannounce

`nannounce(game)`

心跳检测功能让主机出现在游戏列表 (nlist) 中。在主机运行时定时调用该功能。

```
nannounce("duel")
```

nkb

```
nkb([on])
```

地址栏。nkb(1) 打开它，nkb(0) 隐藏它，nkb() 显示它是否打开。上下键切换字符，可以是 0-9、句点或冒号。A 键写入。X 键删除。键盘不会向其中输入任何内容。

```
nkb(1)
```

nkbstr

```
nkbstr([s])
```

地址栏中的文本，然后字符上下移动。nkbstr("") 清除文本。

```
addr, digit = nkbstr()
```

_mp

```
_mp(from, name, ...)
```

你需要编写这个函数。它会在收到消息时运行。`from` 是发件人 ID，`name` 是事件名称，后面最多可以有四个值。

```
function _mp(from, name, a, b)
  if name == "pos" then
    px[from] = a
    py[from] = b
  end
end
```

_mp_join

`_mp_join(p)`

你编写了这个函数。一个对等节点收到了 ID 为 p 的节点。它也会为你触发；whoami() 函数返回的就是那个 ID。

```
function _mp_join(p)
  add(log, p)
end
```

_mp_leave

`_mp_leave(p, why)`

你编写了这个函数。对等节点 p 已离开。为什么 0 表示正常离开。为什么 1 表示连接丢失。

```
function _mp_leave(p, why)
  px[p] = nil
end
```

_mp_update

`_mp_update()`

你编写了这个函数。它每 15 个 _update 帧运行一次，大约 4 Hz。这是发送位置信息的理想位置。

```
function _mp_update()
  nsend("pos", x, y)
end
```

_mp_fail

`_mp_fail()`

你编写了这个函数。会话已断开：主机消失，或者 ncall 调用失败三次。

```
function _mp_fail()
  online = false
end
```

协程

cocreate

```
cocreate(f)
```

从函数 f 创建一个协程。它要等到 coresume 才会运行。

```
co = cocreate(cutscene)
```

coresume

```
coresume(c, ...)
```

运行协程 c 直到它产生值或返回为止。成功时：返回 true 和产生的值。出错时：返回 false 和错误信息。

```
ok, a = coresume(co)
```

costatus

```
costatus(c)
```

“暂停”、“运行中”或“死亡”。

```
if costatus(co) == "dead" then co = nil end
```

yield

```
yield(...)
```

将控制权交还给调用 coresume 的用户。这些值将成为 coresume 的结果。

```
yield(1)
```

措施

info

```
info([x, y, col])
```

_draw 函数用于绘制实时 HUD，参数包括：fps（每秒平均值）、upd、drw、px、max、ram、peak、ramc 和 net（会话启动时）。info() 函数使用打印光标。info(col) 仅设置颜色。info(x, y) 或 info(x, y, col) 设置颜色位置。数值使用 B、k 或 m 单位，以便适应小屏幕。调用本身不计入像素。

```
info(1, 1, 7)
```

showstats

```
showstats([on])
```

与信息显示相同的HUD，在_draw函数执行后自动绘制在(0, 0)位置，忽略相机和剪辑。showstats(0)隐藏它。showstats(1)或showstats()显示它。

```
showstats(1)
```

perf

```
perf(n)
```

一个HUD数字，方便你自行绘制。0帧/秒，1更新，2绘图，3像素，4最大值（当前帧数最高），5内存，6内存+游戏卡带，7游戏卡带启动以来的峰值内存。

```
print(perf(0), 2, 2, 7)
```