# PIX37 Manual

Redigerare, sedan kod. Offline-kopia av boken på pix37.party/docs.

# Redaktör

PIX37 DEV är redigeraren. Du ritar, skriver och lyssnar på spelet här, och exporterar sedan en spelkassett eller en fristående version. Den kan också spela upp en spelkassett, och den kan öppna PIX37.PARTY för att ladda ner spelkassett och använda katalogen. PIX37 DEV EDUCATIONAL är samma redigerare, och den är gratis. Den har inga versioner, ingen export av spelkassett och ingen åtkomst till PIX37.PARTY. PIX37 Console är spelaren. Den spelar upp `.p37` -spelkassett, och du öppnar PIX37.PARTY för att ladda ner spelkassett och bläddra i katalogen.

## Projekt och spelkassetter

En `.pix37` -fil är projektet: källflikar, sprite-ark, kartor, ljud och datatabeller. En `.p37` fil är spelkassett som spelaren kör: tätt packad bytekod.

Sparade filer och prestationer finns i författarmappen bredvid appdata. I Windows ser det ut så här: `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` för en sparning, och `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` för prestationer. I Linux är samma filer `~/.local/share/PIX37/Mira/Pong.p37s` och `~/.local/share/PIX37/Mira/Pong.ach` . Ada är kontot på datorn. Mira är författarnamnet, och det namnet är mappen. Pong är sparplatsen, som som standard är speltiteln. `save` skriver `.p37s` . Den första upplåsta prestationen skriver `.ach` . Om filerna redan finns där läser löparen dem när spelkassetten startar.

## Utvecklarmeny

Escape öppnar utvecklarmenyn. Escape medan ett spel körs återgår till redigeraren. Bekräfta med Enter eller A-knappen (V som standard). Tillbaka är B-knappen (C som standard).

Utan laddat projekt är menyn: SKAPA PROJEKT, LADDA PROJEKT, BYGG, LADDA spelkassett, ANSLUTA TILL PIX37.PARTY, INSTÄLLNINGAR.

Med ett projekt laddat är det: SPARA PROJEKT, LADDA PROJEKT, SKAPA PROJEKT, TILLGÅNGAR, BYGG, LADDA spelkassett, ANSLUTA TILL PIX37.PARTY, INSTÄLLNINGAR.

Ctrl+S sparar det öppna projektet. Ctrl+R kör det.

## Skapa ett projekt

SKAPA PROJEKT frågar efter spelkassetten innan du börjar skapa projektet.

- **VIDEO** — 1-BIT (2 färger), 2-BIT (4), 4-BIT (16), 8-BIT (256).
- **LJUD** — samplingsdjup för synthen: 16, 8, 4, 2 eller 1 bit.
- **SKÄRM** — förinställningarna inkluderar 64×64, 128×128, 160×90, 256×256 och 320×180. Med ANPASSAD storlek kan du välja valfri storlek från 1×1 till 512×512.
- **PALETTERS** — färgbanker. Den första banken börjar som 16-färgsuppsättningen nedan. LOAD PALETTE tillämpar en inbyggd uppsättning: PICO-8 16, PICO-8 32, Game Boy, 1-bit, CGA, Sweetie-16 och en längre lista hämtad från Lospec (Resurrect 64, Apollo, Endesga 32 och 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century och andra). Du kan också ladda en palettfil eller ladda din egen PNG och ta bort paletten från den bilden.




Färgindex som används av `cls` , `pset` , `spr` och resten. Du kan måla om vilket index som helst i palettredigeraren.

## De fem panelerna

Redigeraren är GAME, PAINT, CODE, SOUND och DATA, i den ordningen. Kartan finns inuti PAINT. SFX och låtar finns inuti SOUND. Klicka på en panel för att arbeta där. När CODE har fokus ignorerar SOUND och DATA bokstavstangenter så att skrivning inte också spelar piano.




Fullständig layout, skriven `1:1:4:1:1` . Varje siffra anger hur stor del av raden panelen tar upp: SPEL, MÅL, KOD, LJUD, DATA. En liten skärm sitter centrerad i en cell på 128×128. En skärm större än 128 px breddar SPEL-kolumnen.

INSTÄLLNINGAR → GLOBAL → LAYOUT cyklar arrangemangen:

| Koda | Vad du ser |
|---|---|
| `1:1:4:1:1` | Allt. SPEL och LJUD staplade, MÅL och DATA staplade, KOD till höger. |
| `1:1:4:1:1 2X` | Samma upplägg. KODEN ritas med dubbel densitet, så dubbelt så mycket text får plats. |
| `1:0:1:0:1` | SPEL, DATA, KOD. |
| `1:0:1:0:1 2X` | Samma arrangemang, med den tätare KODEN. |
| `0:1:1:1:1` | Ingen förhandsvisning av spelet. MÅLA, KOD, LJUD, DATA. |
| `0:1:1:1:1 2X` | Samma arrangemang, med den tätare KODEN. |
| `0:1:4:1:2` | Studio. Bred KOD, en högre DATA-remsa, MÅL och LJUD. |
| `0:1:4:1:2 2X` | Samma arrangemang, med den tätare KODEN. |
| `0:0:4:0:4` | Endast KOD och DATA, båda stora. |
| `0:0:4:0:4 2X` | Samma arrangemang, med den tätare KODEN. |
| `0:1:4:1:0` | MÅL, LJUD och en bred KOD. |

| | |
|---|---|
| `0:1:4:1:0 2X` | Samma arrangemang, med den tätare KODEN. |
| `0:4:0:0:0` | Endast MÅL, 256×256, en pixel av arket till en pixel av duken. |

En `2X` rad behåller samma paneler. CODE-panelen ritas med dubbel upplösning och skalas tillbaka till samma cell, så texten, tabbarna och fästringen är hälften så höga och dubbelt så mycket källkod får plats på skärmen.

## SPEL

Förhandsvisningen av den pågående spelkassetten. Klicka på SPELpanelen eller tryck Ctrl+R för att köra förhandsvisningen. INSTÄLLNINGAR → SPELPANEL är där panelen konfigureras:

- **Förhandsgranska** — inaktiv färg, kontinuerlig eller automatisk omladdning när du redigerar.
- **Spela i ett nytt fönster** — spelet öppnas bredvid redigeraren.
- **Visa** — förhandsgranska som RGB888 eller RGB565.
- **SPEL 64** — en 64×64-skärm är antingen centrerad 1:1 i cellen eller skalad 2× för att fylla den.
- **Shaders** — CRT och filter i spelet, separata från filtren i redigeraren.
- **Färg** — panelens bakgrund.

## MÅLA

Sprite-ark och, på kartsidan av samma panel, kakelkartor. Arken kan vara större än skärmen. Förhandsgranskningsstorlekarna för sprite är 8, 16, 32, 64 och 128 (128 endast när arket är minst 128 px). Rulla mushjulet över sprite-förhandsgranskningen för att zooma den. Penselstorlekarna är 1, 2, 3 och 4.

| Nyckel | Verktyg |
|---|---|
| B | Penna |
| M | Rektangelval |
| Q | Lasso |
| S | Stämpel |
| G | Hink |
| Jag | Pipett |
| V | Flytta |
| L / Skift+L | Linje / kurva |
| U / Skift+U | Ihålig rektangel / ihålig ellips |

| | |
|---|---|
| F / Skift+F | Fylld rektangel / fylld ellips |
| D | Dither |
| [ ] | Mindre/större borste |
| Ctrl+Z / Ctrl+Y | Ångra / gör om. Ctrl+Shift+Z gör också om. |
| Ctrl+C / Ctrl+X / Ctrl+V | Kopiera, klipp ut, klistra in |
| Ctrl+A | Välj arbetsytan |

Kartan använder samma idé, med färre verktyg: B penna, M rektangelmarkering, S stämpel, V flytta och Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Håll Alt nedtryckt för att zooma kartans struktur. Penselstorleken på kartan bibehålls med målarverktygen.

Varje sprite har åtta flaggbitar. Paint sätter dem på spriten; `fget` och `fset` läser dem i spelet.

INSTÄLLNINGAR → MÅLREDIGERING är där panelen konfigureras:

- **Palett** — vilken bank som dras, plus ladda och spara.
- **Förhandsvisning** — RGB888 eller RGB565.
- **Penselmarkör** — pekaren medan du målar.
- **Pixel snap** — hur penseln snapar till pixlar.
- **Stor borste** — en stor borste knäpps från sitt övre vänstra hörn, eller från hela sitt fotavtryck.
- **Lökskal** — ett spöke från andra ramar, och vilken riktning det spöket är vänt mot.
- **Kartzoom** — riktiga pixlar, eller en fri zoom.
- **Färger** — färgerna i gränssnittet för målning.

## KODA

Flikar är separata filer inuti projektet. Klicka på ett fliknamn för att byta namn på det; Enter bekräftar. Flera flikar som var och en definierar `_init` , `_update` eller `_draw` är sammanfogade i tabbordning: den första fliken körs först, den sista fliken körs sist. Kod som inte finns inuti en hook blir en del av `_init` . En spelkassett utan några hooks alls behandlas som en `_draw` .

| Nyckel | Handling |
|---|---|
| Pilar | Flytta cirkumflätan. Skift utökar markeringen. |
| Ctrl+Vänster / Höger | Med ord. |
| Hem / Slut, Ctrl+Home / Slut | Början eller slutet av raden, eller av tabulatorn. |

| | |
|---|---|
| Flik | Infoga två mellanslag. Acceptera namnet i kompletteringslistan. |
| Skriva in | Ny rad, indragen som den föregående. Accepterar en komplettering. |
| Ctrl+C / X / V | Kopiera, klipp ut, klistra in. |
| Ctrl+Z / Ctrl+Y | Ångra / gör om. |
| Ctrl+A | Välj fliken. |
| Ctrl+D | Duplicera linjen eller markeringen nedåt. |
| Ctrl+/ | Kommentera eller avkommentera med `--` . |
| Ctrl+F | Hitta. F3 hoppar till nästa träff. Escape stänger ribban. |
| Ctrl+G | Gå till ett radnummer. |
| Fly | Stänger sök, gå till eller ledtrådslistan först. En andra Escape-tangent lämnar redigeraren för utvecklarmenyn. |

Tips dyker upp medan du skriver ett funktionsnamn. Enter eller Tab infogar namnet, och ett `(` om KOMPLETT ( är aktiverat i inställningarna. INSTÄLLNINGAR → KODREDIGERING:

- **Sök** — Ctrl+F i den aktiva fliken, eller över alla flikar.
- **Text hög** — beskär tecken till 5 px, eller visa hela tecknet.
- **Strängstorlek** — teckenstorleken för strängliteraler.
- **Kommentartypsnitt** — 3×5, eller Fusion-typsnittet för det aktuella användargränssnittsspråket.
- **Komplett (** — om komplettering skriver in parentesen.
- **Kodtips** — på eller av.
- **Tipsspråk** — menyspråket, eller engelska.
- **Cirkumflex** — extra förskjutning på tomma rader.
- **Ränna** — radnummer.
- **Färg** — bakgrund, kommentarer, tabbar, siffror, operatorer, nyckelord, strängar, motorfunktioner, dina funktioner, variabler, hakparenteser, cirkumfläta, markering, felrad, varning.

## LJUD

Panelen har två flikar. SFX är ett mönster: en kort serie noter. SOUND är en sång: den placerar dessa mönster på åtta kanaler. TIC är hur många ticks ett steg väntar. Klockan är 120 ticks per sekund, och ett steg är en sextondelsnot, så 15 ticks är 120 slag per minut. En högre TIC är ett långsammare mönster. Två mönster håller takt när en TIC delar den andra: 15 och 30 gör det, 15 och 8 gör det inte.

Pianot spelar när en not väljs med musen. Vita tangenter är ZXCVBNM (CDEFGAB). Svarta tangenter är SDGHJ (C# D# F# G# A#). Håll muspekaren över ett värde och skriv in ett nummer, eller håll muspekaren över det och

skrolla med hjulet. Tonhöjdstal i kod går från 0 (C-0) till 63 (D#5).

På ett mönster är LOOP och LEN samma två siffror. OUT över 0 loopar från IN till OUT, och titeln är LOOP. OUT vid 0 och IN över 0 är LEN: så många toner spelas från början, sedan slutar mönstret, och titeln är LEN. Båda vid 0 spelas varje steg.

Hur mycket av synten du kan redigera beror på vilket bitdjup som valts för spelkassetten. INSTÄLLNINGAR → SFX-REDIGERING ställer in djupet, om en ton automatiskt går framåt (av, tonkedja eller fältkedja), om mellanslag startar om uppspelningen eller växlar mellan start och stopp, samt redigerarens färger.

### 16-bitars

Varje steg har en ton, en volym, en våg och en effekt. Volymen är från 0 till 7. 0 är tyst.

Vågor:

- **0 triangel**
- **1 lutande såg**
- **2 såg**
- **3 kvadrat**
- **4 pulser** — en smal puls
- **5 organ**
- **6 brus**
- **7-fasare**
- **8 PWM** — en kvadrat vars bredd rör sig
- **9 pulser 25%**
- **10 sinus**
- **11 metalliskt ljud**
- **12 FM-klocka**
- **13 plocka**
- **14 supersåg**
- **15 PWM (C64)**

Effekter på en enda ton:

- **0 ingen**
- **1 bild** — tonhöjd och volym glider till nästa ton över detta steg
- **2 vibrato** — tonhöjdsvängningar
- **3 dropp** — tonhöjden tystnar, den vanliga sparken
- **4 tonar in**
- **5 tona ut**
- **6 arpeggio** — går över denna not och de följande tre, en var 8:e tick. Vid TIC 8 eller lägre, var 4:e tick.
- **7 snabba arpeggio** — samma fyra toner, var fjärde tick, eller varannan när TIC är 8 eller lägre.

Mönsterrubriken, ovanför noterna:

- **TIC** — tick per steg, från 1 till 255. 15 är 120 slag per minut. Skriv in numret, eller stega det ett steg i taget.
- **LOOP IN / OUT** — med OUT över 0 loopas mönstret från IN till OUT. Om OUT lämnas vid 0 blir titeln LEN: IN är hur många toner som spelas, och 0 betyder hela mönstret.
- **BRUS** — hela mönstret blir brus.
- **CRUSH** — en livlig variant av varje våg.
- **DTUNE** — 0 av, 1 en lätt refräng, 2 en stark refräng.
- **RVERB** — efterklang: 0 av, 1 kort, 2 långt.
- **DAMPN** — ett lågpassfilter: 0 av, 1 ljus, 2 stark.
- **LEVEL** — volymen för hela mönstret, 0 till 7.
- **SWING** — blanda sig. Udda steg väntar lite längre, jämna steg lite kortare.
- **STEG** — hur många steg mönstret använder: 32, 24, 16 eller 12.

### 8-bitars

Ett steg har en ton och en våg. De fyra vågorna är fyrkant, triangel, såg och brus. Rubriken har TIC, LOOP IN/OUT, FX och LEVEL. TIC är en av 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60. FX är en effekt för hela mönstret: 0 ingen, 1 vibrato, 2 arpeggio, 3 fade out. LEVEL är 0 till 3.

### 4-bitars

Ett steg är bara en ton. Rubriken har TIC, LOOP (på eller av), WAVE (samma fyra vågor) och OCT (vilken oktav noterna sitter i). TIC är en av 4, 6, 8, 10, 12, 15, 20, 30.

### 2-bitars

Ett steg är tomt, en trumma, en träff eller en kross. Huvudet har TIC och BASE. TIC är 8, 15, 16 eller 30. BASE anger hur högt trumman och träffen sitter.

### 1-bit

Ett steg är en anteckning. TIC är 8, 15, 16 eller 30, samma fyra hastigheter som 2-bitars.

### Låtar

Fliken LJUD är den öppna låten. Åtta kolumner är kanalerna. En cell är ett mönster-ID. 255, ritad tom, betyder att kanalen är tyst på den raden. Kolumnen X är ett kommando: fortsätt, stopp, loopstart eller loopslut. Stopp avslutar låten efter raden. Loopslut hoppar tillbaka till den senaste loopstarten.

Ett varv slutar när mönstret längst till vänster med LOOP OUT vid 0 slutar. Mönster längre till höger gör inte det varvet längre: ett längre mönster klipps av, ett kortare börjar om från sitt första steg tills varvet slutar. Ett

mönster med LOOP OUT över 0 loopar mellan LOOP IN och LOOP OUT och avslutar inte varvet. Om varje mönster på varvet loopar, stannar varvet där.

### Genvägar

| Nyckel | Handling |
|---|---|
| Pianotangenter | Skriv en anteckning om det steg som valts med musen. |
| Upp / Ner | Ändra värdet under markören. |
| Utrymme | Spela den öppna SFX, eller den öppna låten från första raden. Starta/stoppa eller starta om, från SFX EDITOR-inställningarna. |
| Ctrl+Z / Ctrl+Y | Ångra / gör om. |
| Ctrl+C / Ctrl+X / Ctrl+V | Kopiera, klipp ut, klistra in markeringen. |

## DATA

Namngivna tabeller. Varje tabell består av rader och kolumner. Från koden används dessa tabeller av `dget` , `dset` , `dstr` , `drows` och `dcols` . Rad- och kolumnnummer i koden börjar på 0.

Kolumntyper, cykliska i rubriken:

- **bool** — en bit.
- **2** — två bitar.
- **4** — fyra bitar.
- **8** — en byte.
- **16** — 16 bitar, för id:n och större tal.
- **spr** — ett sprite-id.
- **str** — text, ritad med systemteckensnittet.
- **str8** och **str10** — samma text, ritad högre.
- **f16** och **f32** — en halvprecisionsflottör och en fullflottör.

Med pekaren på en kolumnrubrik ändrar Ctrl+skroll kolumnens typ. En vanlig skrollning rör sig över kolumner eller nedåt i rader, beroende på vilket INSTÄLLNINGAR → DATAREDIGERING som väljs. Ctrl byter riktning, vilket är anledningen till att Ctrl+skrollning på en rubrik ändrar typen när hjulet är inställt på kolumner.

Piltangenterna flyttar. Enter redigerar en cell. INSTÄLLNINGAR → DATAREDIGERING:

- **Hjul** — rulla kolumner eller rulla rader.
- **Färger** — rutnätet, sidhuvudet och markeringen.

## Tillgångar

ASSETS importerar och exporterar den del som är öppen, inte hela projektet. Sprites använder spriten öppen i PAINT, på arket som är öppet. Kartan är kartan som är öppen. SFX är mönstret som är öppet. Musik är låten som är öppen. Data är tabellen som är öppen. Kod är undantaget: varje tabb skrivs och varje tabb läses tillbaka.

| Slag | Fil | Vad som rör sig |
|---|---|---|
| Sprites | `.png` | Den öppna spriten. |
| Karta | `.p37map` | Den öppna kartan. |
| SFX | `.p37sfx` | Det öppna mönstret. |
| Musik | `.p37mus` | Den öppna låten. Mönsternumren förblir som de är. Exportera SFX separat om du behöver dessa ljud. |
| Data | `.p37dat` | Det öppna bordet. |
| Koda | `.lua` | Varje flik. |

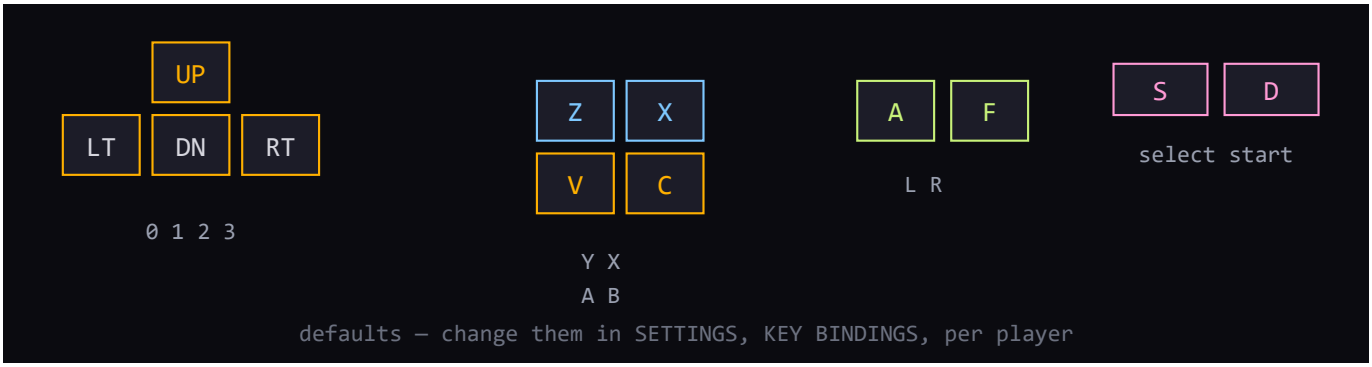
## Inställningar

Rotlistan:

| Sida | Vad den lagrar |
|---|---|
| PROFILER | Spara och ladda alla inställningar. Nytt, byt namn, skriv över, radera. Kopiera eller klistra in profilen som text. |
| Meny | Fönster, tangenter, menyfärger, valfri bakgrund. |
| SPELPANEL | Förhandsgranskning och uppspelningsfönstret. |
| KODREDIGERING | Markeringar och tips. |
| MÅLREDIGERING | Pensel, palett, kartvy. |
| SFX-REDAKTÖR | Synt och notinmatning. |
| DATAREDAKTÖR | Hjulriktning och färger. |
| SPRÅK | Meny- och tipsspråk. |
| LICENSER | Licenstexter, inklusive teckensnittet Fusion Pixel och SDL3. |

GLOBAL:

- **LAYOUT** — panelarrangemang, ovan.
- **VOLYM** — masternivå, 0 till 10.
- **TANGENTBINDNINGAR** — tangenter för var och en av 8 spelare. Tryck på en tangent för att fånga den. Instruktioner, ABXYLR, välj.
- **Presentera** — hoppa över vilolägesbildrutor eller rita varje bildruta.
- **Helskärm** — kantlös.
- **Uppdatera** — fönster-FPS. Vänster och höger steg 60, 90, 120 och max. Enter skriver in ett tal upp till 1000. 0 betyder ingen gräns.
- **Vsync** — vänta på visningen mellan bildrutorna.
- **Skala** — heltalsfönstermultiplikator.
- **Shaders** — CRT och filter i redigeraren, inte i spelet.
- **Färg** — menytext och markering.
- **Bakgrund** — en valfri PNG, beskuren till 128, bakom menyn. Rensa tar bort den.
- **Redigerarmarkör** — pekarform.
- **Mus** — flytande pekare, eller fäst till pixlar.
- **UI-hastighet** — hur snabbt en hållen riktning upprepas genom en lista.
- **Utforskaren** — fillistor börjar vid datorns rot eller i exe-mappen.




Standardtangentbord. A är bekräfta i menyer. B är tillbaka. I ett spel är dessa knappar 4 och 5. Välj (S) är knapp 10, inte en menygangväg.

| Knapp | Roll | Standardnyckel |
|---|---|---|
| 0 | Vänster | Vänsterpil |
| 1 | Rätt | Högerpil |
| 2 | Upp | Uppåtpil |
| 3 | Ner | Nedåtpil |
| 4 | A, bekräfta | V |
| 5 | B, rygg | C |

| | | |
|---|---|---|
| 6 | X | X |
| 7 | Y | Z |
| 8 | R | F |
| 9 | L | En |
| 10 | Välja | S |
| 11 | Start | D |

## Bygga

Välj en plattform: Windows, Linux, Mac, Android, iOS eller Web.

- **Namn** — filnamnet på bygget.
- **Författare** — på Windows och Linux. Detta är mappnamnet för sparade filer.
- **Steam-app-ID** — på Windows och Linux. Tomt betyder att Steamworks inte fungerar. Ett ifyllt id kopplar `achieve(0)` … `achieve(15)` till Steam-achievements ACH_0…ACH_15.
- **Id** — paketidentifierare där plattformen frågar efter en.
- **Dölj kod** — på dator- och mobilöverlägg. Det gör att spelkassetten inte `peek` så fort som möjligt. Det är inte ett lås.
- **Helskärm** — den exporterade spelaren startar i helskärm.
- **Shaders** — upp till tre lager inbyggda i spelaren.
- **Ladda ner .p37** — Endast webben. Visar en nedladdningspost i pausmenyn.
- **Ikon** — en rutmall från ett sprite-ark: ark, storlek, x, y. Eller ett PNG-omslag som du beskär.
- **Exportera** — välj mappen. Menyn visar BYGG SPARAD eller BYGG MISSLYCKADES.

Skapa en build med internet påslaget. Splash-funktionen kan anonymt fråga servern om den här buildens hash är känd. Den kontrollen finns där av två anledningar. Den skyddar mig, författaren till PIX37, från kopior av verktyget som inte var mina att ge bort. Den skyddar också personen som skapade spelet. Att ta någon annans spel och lista det som sitt eget blir allt vanligare på telefonmarknaden, till exempel Play (Android) eller iOS. Spelet startar fortfarande. Om maskinen är offline säger introt att spelet inte är anslutet. Den raden är också en ledtråd till spelaren om att kopian kanske inte kommer från en laglig källa. Det är en liten kontroll, så att om ditt spel stals och såldes utan ditt samtycke av någon annan, kan du göra det i en butik som Play (Android), Steam eller itch.io. Servern lagrar den tidpunkt då hashen först registrerades, bredvid hashen, och ändrar den inte senare. Den tidigare tidpunkten är den ursprungliga builden. En hash som registrerats efter det är en senare kopia.

Jag vill att den här kontrollen ska skydda de som skapar spel. Spelmotorn gör redan mycket av jobbet åt dem, och det här är en av de sakerna. Jag förstår att det kan vara kontroversiellt. Om communityn bestämmer sig för att det är bättre att slopa den här mikro-DRM-funktionen, så kommer jag att göra det.

En gammal spelkassett från en tidigare editor kan vägra att starta efter att språket har ändrats. Öppna `.pix37` -projektet i den här editorn och exportera en ny `.p37` .

### Anslut till PIX37.PARTY

ANSLUT TILL PIX37.PARTY finns i appen. Det finns ingen inloggningssida i webbläsaren. Du loggar in med en Steam-sessionsbiljett eller med itch.io med hjälp av surfplattan och musen. Tjänsten visar ett unikt visningsnamn, 2 till 16 tecken långt. Licensnyckeln och Steam-ID:t visas inte för andra spelare.

Efter användarvillkoren är navet Favoriter, Katalog, Rankning, Spelare och din profil.

En utvecklarlicens är redigeraren. Du skapar spelet, du laddar upp det och du lämnar stjärnor. En spellicens är spelaren. Hjärtan kommer från personer som spelar på en runner.

- **Katalog** — bläddra bland spel. Filter täcker genre (upp till tre), skärmstorlek, spelkassettstorlek och om uppladdningen innehåller ett `.pix37` projekt.
- **Favoriter** — spel du markerat.
- **Uppladdning** — kräver en utvecklarlicens. Du skickar en `.p37` fil och kan bifoga `.pix37` -projektet. En `.p37` är packad bytekod. Den är svår att läsa och kan avsiktligt förvrängas. En `.pix37` fil är projektfilen du faktiskt arbetar i: källkoden, bilderna och ljudet, i en form som en annan person kan öppna och lära sig av. Genom att skicka den kan andra skapare studera spelet istället för en komprimerad spelkassett. Titel, författare, genrer, omslag, ikon och prestationsnamn följer med. RAM-numret som skickas upp är den topp som spelkassett nådde.
- **Kommentarer och prestationer** — på spelsidan. Att låsa upp en prestation medan du är inloggad lagras på det kontot såväl som på maskinen.
- **Rankning** — utvecklare, sorterade efter stjärnor eller hjärter. Varje rad visar båda siffrorna. Öppna ett namn för att se den författarens spel, sorterade efter hjärter.
- **Spelare** — konton sorterade efter upplåsta prestationer.

# Koda

Språket är ett litet skript i Lua-stil. Den här halvan utgår från ett tomt projekt, lär ut språket och visar sedan ett kort exempel på varje funktion.

### Ett första program

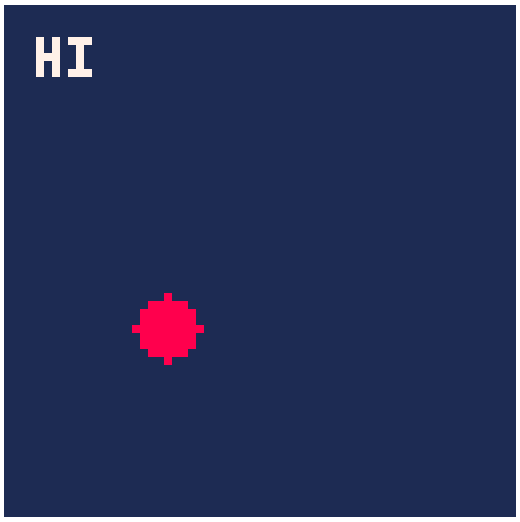
Skapa ett projekt med 64×64, 4-bitars färg. Öppna CODE. Ersätt fliken med:

```
_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
```

Ctrl+R kör den. Piltangenterna flyttar cirkeln. Escape stoppar spelet och återgår till redigeraren.




Samma tre anrop: `cls(1)` målar den marinblå skärmen, `print` ritar HI och `circfill` ritar den rosa cirkeln.

## Vad som körs, och när

- `_init` körs en gång.
- `_update` körs med 60 Hz. Placera rörelse och knappar här.
- `_draw` målar. Utan fps-argument målar den så snabbt som datorn och skärmen kan. `_draw(30)` begränsar målningshastigheten till 30 fps och lämnar logiken på 60.
- `_set_fps(n)` ändrar den gränsen senare. 0 rensar den.

Namn är inte skiftlägeskänsliga; editorn lagrar dem med gemener. `function` och `fun` är samma nyckelord.

## Språk

### Grunderna i programmeringen

Ett program är en lista med rader som spelkassetten kör. Varje rubrik visar först raden, sedan ett litet exempel. En rad som börjar med `--` är en kommentar: en anteckning till dig. Spelet hoppar över den.

#### Variabler

En variabel är ett namn som innehåller ett värde. Du väljer namnet. Senare rader kan läsa det och de kan lagra ett nytt värde med samma namn.

Formen är `name = value` . `local name = value` håller namnet inuti det aktuella blocket, så resten av spelkassetten kan inte se det. Utan `local` är namnet globalt och alla flikar kan se det.

```
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` betyder att det inte finns något värde. `true` och `false` är yes och no. Ett namn du aldrig har angett läses som `nil` .

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

#### Tabeller

En tabell innehåller många värden under ett namn. En lista använder siffror. Det första elementet är 1, inte 0. En post använder namn. En tabell kan innehålla en annan tabell.

Listform: `{ value, value }` . Postform: `{ name = value, name = value }` . En lista med poster är en tabell vars objekt är tabeller.

```
-- 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
```

### Kondition och kropp

Ett villkor är en fråga med ett ja- eller nej-svar, såsom `hp > 0` eller `btnd(4)` . En kropp är den grupp av linjer som löper när du går in i ett block. `if` och every loop använder samma form: ett villkor, sedan en kropp, sedan `end` . `repeat` avslutas med `until` istället för `end` .

### Om

`if` kör sin text endast när villkoret är sant. `elseif` försöker ett annat villkor. `else` körs när alla villkor var falskt.

Form: `if condition then` , då kroppen, då `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
```

### Loopar

En loop upprepar en kropp. `while` kontrollerar villkoret först och fortsätter så länge det förblir sant. `repeat` kör kroppen och kontrollerar sedan, så kroppen körs alltid minst en gång. `for` counts. Det sista talet inkluderas. Utan ett tredje tal är steget 1, så om början redan är förbi slutet körs inte kroppen. Nedräkning behöver ett negativt steg.

```
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
```

### Går vid ett bord

`all` besöker varje objekt i en lista, från det första. `pairs` besöker varje namn och värdet som lagras under det namnet.

```
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
```

### Funktioner

En funktion är ett namngivet recept. Formen är `function name(arguments)` , sedan en brödtext och sedan `end` . Du anropar den genom att skriva namnet och parenteser. `return` skickar ett värde tillbaka, och den kan skicka mer än ett. Argumenten är lokala värden för den funktionen.

```
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` är en funktion som kan se de lokala värdena runt omkring sig. Det är en closure. En tabell kan innehålla en, och du anropar den med en punkt.

```
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
```

### Ja, nej, och att välja

vänder `not` ett villkor. `and` är sant endast när båda sidor är sanna. `or` är sant när endera sidan är sann. `a and b or c` väljer `b` när `a` är sant, annars `c` . `cond ? a : b` är samma idé på en rad: `a` när villkoret är sant, `b` när det inte är det.

```
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)
```

### Matematik och text

`+ - * /` är addera, subtrahera, multiplicera och dividera. `//` dividerar och behåller hela talet. `%` är resten. `^` är potens. `..` limmar ihop två texter. `#t` är antalet objekt som finns i en lista. Jämförelser är `==` (samma), `~=` eller `!=` (olika), och `< > <= >=` . `x += 1` adderar 1 med x. Samma form fungerar för `-= *= /= ^= %=` .

```
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
```

### Där en lokalbor bor

`local` varar till `end` som stänger dess block. `if` / `end` är dess eget block. Du kan nämna en local ovanför raden som sätter den, men tills den raden körs är värdet `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
```

### Hoppning

`::label::` markerar en linje. `goto label` hoppar dit, men bara inom samma funktion.

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

## Minne, kort sagt

`peek` och `poke` på levande pixlar genom en 16-bitarsadress. De lagrar inte en andra kopia av spelkassetten. Adresserna `0x5F00` till `0x5FFF` är register: kamera, klipp (skrivskyddad), ritfärg, fyllningsmönster, de tre palettmappningarna och sidnumren. Bytena på båda sidor om det hålet, 65280 av dem, är pixelfönstret.

När skärmen får plats med de 65280 byte, 64×64 och 128×128 gör det och 256×256 inte gör det, börjar skärmen vid adress 0, rad för rad. Det som är kvar är sprite-arket som valts av `0x5F41` (0 är ark 0). Om det arket är större än det återstående utrymmet väljer `0x5F42` sidan och kartan visas inte i fönstret. Om arket får plats är svansen kartceller från kartan som valts av `0x5F43` , en eller två byte vardera, och `0x5F44` sidor kartan när den inte får plats i svansen.

När skärmen inte får plats är hela fönstret en sida på 65280 pixlar. `poke(0x5F40, n)` vänder skärmens sida. `poke(0x5F41, n)` med n annat än 0 visar ark n istället, och ark 0 har ingen vy i detta läge. `poke(0x5F43, n)` med n annat än 0 visar karta n, och det vinner över arket. Karta 0 i detta läge är endast `mget` och `mset` .

En `peek` som inte landar på en pixel eller ett register returnerar 0. En `poke` där gör ingenting. Sprite-flaggor, SFX och DATA-tabeller finns inte i det här fönstret. Använd `fget` , `sfxget` och `dget` .

## Praktisk användning

Varje funktion listas längst ner på den här sidan, med praktiska användningsområden för PIX37-funktioner. Länkarna under varje exempel öppnar mer information om funktionerna.



Smidig rörelse

En spelkassett på 128×128, sedd ovanifrån. Sprite 1 är redan ritad. Pilar accelererar i alla riktningar, och friktion saktar ner dig. Sprite lindas: en halva som lämnar botten ritas upptill, och detsamma gäller för de andra kanterna.

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



## Kontrollerar om nästa drag är blockerat

En 64×64 spelkassett. Golvet och den högra väggen finns redan på kartan, målade med sprite 3, och flagga 0 är satt på den sprite. Pilar flyttar sprite 1. Ordet högst upp säger blockerad när nästa steg skulle överlappa en hel platta, och fri när det inte skulle göra det.

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



## En våtservett med flip

Tryck på A. Wipe-funktionen ritas med flip(), en svart rad i taget, istället för inuti _draw(). _draw förblir ur vägen medan det körs, så varje rad visas på skärmen.

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



## En tabell med funktioner (klass)

Sprite 1 är redan ritad. make_monster fungerar som en klass. Den returnerar en tabell, och update och draw är metoder som behåller monstrets egna x och y. Siffran högst upp visar hur långt det har gått.

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



## Fyra sätt att läsa en knapp

A är knapp 4. Håll den intryckt och laddningen stiger. Släpp den och laddningen går tillbaka till 0. Upp och ner upprepas medan den hålls intryckt, så att menyraden fortsätter att röra sig. btnd är den första bildrutan i ett tryck, vilket denna spelkassett endast använder som början på det trycket.

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



## Skjut med musen

Sprite 1 är redan ritad i mitten. En vit prick är markören. Vänsterklick avfyrar en pixel mot den. Hjulet ändrar hastigheten, tryckt i hörnet.

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



## Skriv ett namn och bekräfta det

Tryck A för att öppna fältet. Upp och ner ändrar en blinkande bokstav. A skriver den, X raderar, Y är mellanslag och B bekräftar. Raden säger sedan ditt namn.

```
btnd keyon keystr flr time cls print _update _draw
```



## Kamera på spelaren

En spelkassett med 128×128-format. Sprite 1 startar mitt på ett golv av kakelplattor. Pilarna rör sig. Kameran håller spriten centrerad. x och y i hörnet rullar inte, eftersom camera() utan argument återställer vyn för HUD:n.

```
btn screen camera cls map spr print flr _update _draw
```



## Några rader ur en visuell roman

Linjerna är i DATA, på engelska och polska. A fortsätter, eller väljer stannar när ett alternativ visas på skärmen. B väljer att lämna ett alternativ, och byter annars språk. X byter språk medan ett alternativ visas. Dialogen använder teckensnittet 8px så polska bokstäver ritas.

```
dstr dget btnd cls print8 print _update _draw
```



## Måla med titt och peta

En spelkassett med 64×64-format, så adress 0 är den övre vänstra pixeln och nästa rad börjar på 64. Spelkassett skriver pixlar, kopierar en rad, rensar en rad och skriver ut de värden den läste tillbaka.

```
cls poke peek poke2 peek2 poke4 peek4 memcpy memset print _draw
```



## En lista du kan lägga till i

A adderar ett tal, upp till 8. B tar bort det sista. X tar bort det första. Listan skrivs ut på skärmen.

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



## Ett namn som tagits isär

Spelkassetten delar ada,12,blå och skriver ut namnet, dess första bokstav, 12, och ordet blå.

```
cls split sub chr ord tonum tostr print _draw
```



## Spela en ton och ändra den sedan

Mönster 0 är en ton långt, och varje uppspelning har samma volym. A spelar det igen när den tonen är slut. B höjer tonhöjden och spelar den tonen.

`sfxset sfxhdr sfx btnd sfxstat sfxget cls print _init _update _draw`



## En låt skriven i spelkassetten

Två mönster loopas på rad 0, ett på var och en av de två första kanalerna. Raden förblir 0. ch0 och ch1 är tonen som varje kanal är på. Att redigera mönstret, LEN och LOOP i spelkassetten är ett sätt att se vad dessa kontroller gör i musikpanelen.

`songget songset music musicstat cls print _init _draw`



## Tre monster i DATA

DATA-panelen har redan en namnkolumn och en hp-kolumn, med slem, fladdermus och öga. A skadar slemmet. Siffrorna till höger är dess hp.

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



## Kom ihåg poängen

A lägger till en poäng och sparar den. Avsluta och öppna spelkassetten igen: numret finns fortfarande kvar. B glömmer det.

`load btnd save delsave cls print _init _update _draw`




## Ett andra kapitel i en annan fil

Två spelkassetter delar på sparuppdraget. Del 1: A markerar spelet som avslutat. Del 2, som öppnas först, säger att del 1 ska avslutas. Därefter öppnas del 2 och B raderar sparningen så att del 1 kan spelas igen.

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



## En prestation

Sprite 8 är redan dragen. Tryck på A tio gånger. Skålen namnger prestationen, och sprite 8 stannar kvar eftersom den är upplåst. B rensar prestationen i den lokala filen och nollställer räknaren.

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



## Ett rum man kan gå in i

En spelkassett på 256×256. En vit prick är markören. Menyn är värd, gå med eller serverlistan. Värden öppnar ett rum med namnet gå och startar spelet. Gå med skrollar fram en adress (siffror, en punkt, ett kolon) och B ansluter. Serverlistan flyttas upp och ner, och A går med i raden. I spelet går pilarna mot sprite 1. Ett mynt delas: värden flyttar det, och om man plockar upp det lägger man till 1 till spelarens poäng och nästa skapas. Varje sprite, inklusive din, visar poängen bredvid. Hörnet visar hur många spelare som är med, LAN-adressen

och värdens publika adress. B lämnar: du och de andra återgår till menyn, och nästa värd startar ett nytt rum med poäng 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`



### Dialog som väntar

A visar nästa rad. Funktionen pausar vid varje utgång tills du trycker på A igen.

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



### Vänta en halv sekund

Ordet ändras från ... till hej efter ungefär en halv sekund. Skärmen fortsätter att rita medan korrutinen är pausad.

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



### Cirklar som anländer en efter en

Fem cirklar droppar in från toppen, två bildrutor ifrån varandra, och faller sedan.

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



### Två bilder, från klockan

Sprite 1 och sprite 2 är två bildrutor av samma karaktär. Bilden växlar fyra gånger per sekund.

`cls` `flr` `time` `spr` `_draw`

## Varje funktion

Argument inom hakparenteser är valfria. Index för tabeller och strängar börjar vid 1, förutom DATA-rader och -kolumner, som börjar vid 0. Färger är palettindex.

### Index

`_rita`
`_init`
`_mp`
`_mp_fail`
`_mp_join`
`_mp_leave`
`_mp_update`
`_set_fps`
`_uppdatering`
`abs`
`achclear`
`achdef`
`uppnå`
`uppnåtts`
`lägga till`
`alla`
`hävda`
`atan2`

band
bnot
bor
btn
btnd
btnp
btnu
bxor
kamera
tak
chr
circ
cirkulera
klickl
klickm
klicker
klipp
cls
samskapa
färg
corsume
cos
costatus
räkna
markören
dcols
del
delikatessbutik
delsave
dget
drunknar
dset
dstr
fget
fyllp
vända
flr
föreach
fset
info
ipairs
avstängning
tangenten
tangentstat
keystr
linje
ladda
log
lshr
karta
max
memcpy
memset
mget
mitten
min
mus
mset
musik
musikstat
tillkännage
ncall
nästa
nhost
njoin
nkb
nkbstr
nlan
lämna
nlist
nlyssna
notera
nspelare
nsend
nstun
ord
oval
ovalfyllning
packa
par
kompis
palrgb
palt
pcall
kika
titt2
titt4
perf
pget
peta
peta2
peta4
skriva ut
tryck 10
tryck 8
pset
rect
rectfill
rnd
rotl
rotr
rätt
rectfill
spara
sparaslot
skärmen
rulla
välj
setmap
setpalett
setspr
settex
vända
sfx
sfxget
sfxhdr
sfxset
sfxstat
sget
sgn
shl
showstats
shr

| | | |
|---|---|---|
| synd | srand | tline |
| songget | sset | tonum |
| sångset | sspr | tostr |
| split | stat | packa upp |
| spr | sub | whoami |
| sqrt | tid | avkastning |

## Livscykel

### _init

`_init() ... end`

Körs en gång när spelkassetten startar. Kod som sitter utanför någon hook foldas också in i _init.

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

### _update

`_update() ... end`

Spellogik. Anropas vid 60 Hz, även när _draw är begränsat eller värden är långsammare. Läs knappar här.

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

## _draw

```
_draw([fps]) ... end
```

Ritar en bildruta. Utan argument, eller 0, körs den så ofta som värden presenterar. Skicka n (1..999) för att begränsa ritningen till n bildrutor per sekund. _update stannar vid 60 Hz.

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

## _set_fps

```
_set_fps(n)
```

Ändrar _draw-taket medan spelet körs. 0 betyder obegränsat. n är begränsat till 0..999.

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

## flip

```
flip()
```

Presenterar skärmen och ger efter tills nästa bildruta. Använd den bara om du ritar din egen bildruteslinga. Vänta inte efter den.

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

## time

```
time()
```

Sekunder sedan _init.

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

## Input

### btn

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

1 medan knapp b hålls intryckt, annars 0. p är spelaren, 0..7. Utan argument returnerar den en bitmask för spelare 0. Knappar: 0 vänster, 1 höger, 2 upp, 3 ner, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 select, 11 start.

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

### btnd

`btnd(b, [p])`

Sant endast på _update-ramen när knapp b trycks ned.

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

### btnp

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

Sant vid tryckning, sedan igen medan den hålls intryckt: 30 tick fördröjning, sedan var 8:e tick. Utan någon knapp returnerar den en mask.

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

### btnu

`btnu(b, [p])`

Sant endast på ramen när knappen släpps.

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

## mouse

```
mouse()
```

Returnerar musens x, y i spelpixlar.

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

## clickl

```
clickl([p])
```

Vänster musknapp. clickl() hålls nedtryckt. clickl(1) är presskanten.

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

## clickr

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

Höger musknapp. Samma kant med ett argument som vid klick.

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

## clickm

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

Mittknappen eller ett klick med rullhjulet.

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

## scroll

`scroll()`

Mushjulets rörelse i den här rutan. Positivt och negativt beror på hjulets riktning.

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

## keyon

`keyon()`

Öppnar ett textfält som styrs av knappar. Upp och ner ändrar en bokstav från A till Ö. A skriver den bokstaven. X raderar. Y är mellanslag. B bekräftar och stänger fältet. Tangentbordet skriver inte i det.

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

## keyoff

`keyoff()`

Döljer textfältet.

```
keyoff()
```

## keystr

`keystr([s])`

Text i fältet, sedan 1 på ramen B bekräftar, sedan ändras bokstaven upp och ner. Den bokstaven är tom när fältet stängs. keystr("") rensar texten.

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

### keystat

```
keystat()
```

1 medan textfältet är öppet, annars 0.

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

## Ritning

### screen

```
screen()
```

Aktuell skärmstorlek för spelkassett i pixlar. Returnerar bredd och sedan höjd. Detta är spelets upplösning, inte redigeringsfönstret.

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

### cls

```
cls([col])
```

Rensar skärmen. Standardfärgen är 0.

```
cls(1)
```

### color

```
color(col)
```

Anger ritfärgen som används när ett senare anrop utelämnar färg. Utan argument returnerar den aktuell färg.

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

### camera

```
camera([x, y])
```

Offseterar varje dragningsanrop. camera() återställs till 0, 0.

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

### clip

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

Begränsar ritningen till en rektangel. Ett femte argument, skilt från noll, skär det aktuella klippet. clip() återställer helskärmen. Peek av klippregistren är skrivskyddat.

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

### cursor

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

Anger var nästa utskrift utan koordinater ska, och valfritt returnerar color. cursor() x, y.

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

### pset

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

Ställer in en skärmpixel. Räknas mot pixelstatistiken.

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

### pget

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

Färg på skärmpixeln vid x, y.

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

### print

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

Ritar 3×5-text (UTF-8, beskuren från Fusion-teckensnittet). Utan x, y skrivs den ut vid markören och flyttas till nästa rad.

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

### log

```
log(...)
```

Lägger till argumenten, separerade med mellanslag, i en textbuffert. pix37_run skriver ut den buffringen vid stdout. Den ritar inte.

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

### print8

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

Samma som för utskrift, med teckensnittet Fusion på 8 px.

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

### print10

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

Samma som för utskrift, med teckensnittet Fusion på 10 px.

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

### line

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

Linje från x0,y0 till x1,y1. Två argument fortsätter från föregående ände. line() återställer den kedjan.

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

### circ

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

Cirkelkontur. Radien är i pixlar.

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

### circfill

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

Fylld cirkel.

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

### oval

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

Ellipsformad kontur inuti avgränsningsramen. Hörnen är inkluderande.

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

### ovalfill

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

Fylld ellips i en avgränsningsram.

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

### rect

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

Rektangelkontur. Hörnen är inkluderande.

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

### rectfill

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

Fylld rektangel.

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

### rrect

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

Rundad rektangelkontur. x och y är det övre vänstra värdet. w och h är storleken i pixlar; båda måste vara större än 0. r är hörnradien och standardvärdet är 0. Den är fastklämd till hälften av den kortare sidan.

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

### rrectfill

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

Fylld rundad rektangel. Samma argument som rrect.

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

### fillp

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

16-bitars fyllnadsmönster för senare fyllda former. 0 är fast. fillp() returnerar det aktuella mönstret. Bitarna följer 4×4-mönstret.

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

### pal

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

Ombildar färg c0 till c1. läge 0 är ritpaletten, 1 visningspaletten, 2 fyllnadsmönsterpaletten. pal() återställer.

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

### palrgb

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

Ställer in en palettpost till RGB, varje kanal 0..255.

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

### palt

`palt([col, t])`

Genomskinlighet. palt(col) eller palt(col, true) gör col transparent. palt(col, false) gör den heldragen. palt() gör varje färg ogenomskinlig.

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

### setpalette

`setpalette(n)`

Väljer palettbank n. setpalette() returnerar den aktuella banken.

```
setpalette(1)
```

### tline

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

Ritar en linje på skärmen medan kartans pixlar samplas från kartpositionen mx, my, mdx, mdy. För kartans exempel stegvis längs linjen.

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

## Sprites och kartor

### spr

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

Ritar sprite n vid x, y. w och h är storleken i pixlar, standard 8. fx och fy vänds horisontellt och vertikalt när de inte är noll. tex är sprite-arkets fliken. rot är vinkeln i vändningar (0..1), samma riktning som sin och cos. 0 lämnar spriten oroterad.

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

### sspr

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

Kopierar en rektangel av sprite-arket till skärmen, skalad till dw med dh. Standardvärdena behåller källstorleken.

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

### sget

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

Färgen på en texel på sprite-arket.

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

### sset

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

Skriver en texel på sprite-arket.

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

### fget

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

Sprite-flaggor. fget(n) returnerar flaggbyten. fget(n, f) är sant när flaggbiten f(0..7) är satt.

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

### fset

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

Sätter sprite-flaggor. fset(n, v) skriver hela byten. fset(n, f, v) sätter en bit.

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

### setspr

```
setspr(n)
```

Väljer det aktiva sprite-arket. setspr() returnerar det aktuella arket. Samma bank som settex.

```
setspr(1)
```

### settex

```
settex(n)
```

Alias för setspr.

```
settex(0)
```

### map

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

Ritar kartan. Cell cx, cy på kartan ritas på skärm sx, sy. cw, ch av 0 betyder hela kartan. Lagret accepteras och ignoreras. tex är kakelarket.

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

### mget

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

Sprite-id i kartcellen cx, cy.

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

### mset

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

Skriver sprite n i kartcellen cx, cy.

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

### setmap

```
setmap(n)
```

Väljer karta n. setmap() returnerar den aktuella kartan. Karta 0 är en verklig karta.

```
setmap(1)
```

## Minne

### peek

```
peek(addr)
```

Läser en byte av det 64 KB stora fönstret. Fönstret är den aktiva skärmen, sedan spritepixlar, och mappar sedan celler när skärmen får plats med 65280 byte. Större skärmar pagineras med poke(0x5F40, page). Adresser som inte är mappade returnerar 0. Skrivs även @addr.

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

### poke

`poke(addr, ...)`

Skriver byte till addr. Extra argument skriver följande byte. Omappade adresser ignoreras. Skrivs också till $addr, val. Att peta på skärmen räknas som en pixelskrivning.

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

### peek2

`peek2(addr)`

Osignerat 16-bitarsvärde, little-endian, vid adress.

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

### poke2

`poke2(addr, val)`

Skriver ett osignerat 16-bitarsvärde, little-endian.

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

### peek4

`peek4(addr)`

Osignerat 32-bitarsvärde, little-endian.

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

### poke4

```
poke4(addr, val)
```

Skriver ett osignerat 32-bitarsvärde, little-endian.

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

### memcpy

```
memcpy(dst, src, len)
```

Kopierar len byte inuti samma 64 KB-fönster.

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

### memset

```
memset(dest, val, len)
```

Fyller len-byte med början vid dest med val.

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

## Matematik

### abs

```
abs(x)
```

Absolut värde.

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

### flr

`flr(x)`

Avrundar mot minus oändligheten.

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

### ceil

`ceil(x)`

Avrundar mot plus oändligheten.

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

### max

`max(a, b)`

Större av de två värdena.

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

### min

`min(a, b)`

Det lägre av de två värdena.

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

### mid

`mid(a, b, c)`

Mitten av de tre värdena. Användbar som klämma.

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

### sgn

`sgn(x)`

Tecken för x: -1, 0 eller 1.

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

### sqrt

`sqrt(x)`

Kvadratrot. Ett negativt argument är inte ett tal.

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

### sin

`sin(x)`

Sinus. x är i tur och ordning: 0, 0,25, 0,5, 0,75 går runt cirkeln en gång. Inte radianer.

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

### cos

`cos(x)`

Cosinus, i sin tur, samma konvention som sinus.

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

### atan2

`atan2(dx, dy)`

Vinkeln för vektorn dx, dy, i kurvor från 0 till 1.

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

### rnd

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

Slumptal i [0, x). Standardvärdet x är 1, så rnd() är 0..1. För en tärning, flr(rnd(6))+1.

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

### srand

```
srand(seed)
```

Sätter ihop slumpgeneratorn så att en senare rnd-sekvens kan upprepas.

```
srand(1)
```

### band

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

Bitvis OCH. Skrivs även a & b.

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

### bor

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

Bitvis ELLER. Skrivs även a | b.

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

### bxor

```
bxor(a, b)
```

Bitvis XOR. Även skriven a ^^ b.

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

### bnot

`bnot(a)`

Bitvis INTE. Skrivs även ~a.

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

### shl

`shl(a, n)`

Skift åt vänster. Skrivs även som << n.

### shr

`shr(a, n)`

Aritmetisk förskjutning åt höger. Skrivs även som >> n.

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

### lshr

`lshr(a, n)`

Logisk förskjutning åt höger. Skrivs även som >>> n.

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

### rotl

`rotl(a, n)`

Rotera bitar åt vänster. Skrivs även som <<> n.

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

### rotr

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

Rotera bitar åt höger. Skrivs även som >><< n.

## Tabeller och värden

### add

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

Lägger till v i tabell t. Om i ges, infogas vid det 1-baserade indexet.

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

### del

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

Tar bort den första posten lika med v.

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

### deli

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

Tar bort index i och returnerar det värdet. Utan i tas den sista posten bort.

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

### all

```
all(t)
```

Iterator över arraydelen, i ordning. Hål stoppar vandringen åt det håll som en 1-baserad array slutar.

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

### foreach

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

Anropar fn(v) för varje arrayelement.

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

### pairs

```
pairs(t)
```

Iterator över varje nyckel och värde, inklusive namngivna fält.

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

### ipairs

```
ipairs(t)
```

Iterator över arraydelen med det 1-baserade indexet.

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

### next

```
next(t, [k])
```

Returnerar nästa nyckel och värde efter k. next(t) börjar i början. Används av par.

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

### count

```
count(t, [v])
```

Längd på arraydelen. Om v anges, räknas hur många poster som är lika med v.

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

### pack

```
pack(...)
```

Packar argumenten i en ny tabell.

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

### unpack

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

Returnerar t[i] till t[j] som separata värden. Standardvärdena täcker hela arraydelen.

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

### select

```
select(i, ...)
```

select("#", ...) returnerar hur många argument som följer. select(i, ...) returnerar argumenten från i och framåt.

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

### assert

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

Avbryts med ett felmeddelande när c är falskt eller nil. msg ersätter standardtexten.

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

### pcall

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

Anropar f. Vid framgång returnerar `true` och resultaten. Vid fel returnerar `false` och feltexten.

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

## Strängar

### chr

```
chr(n)
```

Ett tecken från en bytekod 0..255.

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

### ord

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

Bytekoder för s från index i till j. Index är 1-baserade. Standardvärdet är det första tecknet.

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

### sub

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

Delsträng från i till j, 1-baserad. Uteslut j för att ta svansen.

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

### split

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

Delar upp s i en tabell. sep är som standard komma. Ett tal sep är en segmentbredd. conv konverterar stycken med tonum när det är sant.

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

### tonum

```
tonum(s, [base])
```

Tolkar ett tal. Basen kan vara 2..36. Returnerar noll när texten inte är ett tal.

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

### tostr

```
tostr(v, [digits])
```

Omvandlar ett värde till text. siffror anger decimaler för tal.

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

## Audio

## sfx

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

Spelar mönster n och returnerar kanalen. Utesluter man ch, eller skickar -1, väljs den första lediga kanalen. ch -2 stoppar mönster n på varje kanal. n -1 stoppar en kanal, eller varje kanal när ch utelämnas. n -2 låter ett loopmönster avslutas istället för att loopas. sfx() utan argument returnerar -1.

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

## sfxget

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

Returnerar en tabell med ett mönstersteg: [1] tonhöjd, [2] våg, [3] volym, [4] effekt. Tonhöjden är 0..63 (C-0 till D#5). Vågen är 0..15. Volymen är 0..7 och 0 är tyst. Effekten är 0..7.

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

## sfxset

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

Skriver ett mönstersteg. Alla sex argument krävs.

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

## sfxhdr

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

Mönsterrubrik. sfxhdr(n) returnerar rubriktabellen. sfxhdr(n, key, val) anger ett fält: speed, loop_in, loop_out, level, noise, crush, dtune, rverb, dampn, swing, steps. Talen 0..10 väljer samma fält i den ordningen. speed är TIC, tick per steg. Högre är långsammare. 15 är 120 slag per minut. 16-bitars accepterar 1..255. Smalare syntar fäster vid sin TIC-lista.

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

## sfxstat

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

Mönster-ID som spelas på kanal ch, eller -1. Standardkanalen är 0.

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

## note

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

Spelar en ton utan mönster. Tonhöjd 0..63, våglängd 0..15, volym 0..7, effekt 0..7. Tick är hur länge den håller, i samma 120 Hz-tick som TIC. Utelämnar du tick varar den i 15, ett standardsteg. Effekten löper över den tiden, sedan tystnar kanalen.

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

## music

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

Startar låt n från toppen. music(-1) stoppar. Om man anropar music med det ID som redan spelas lämnas den igång. music() returnerar det spelade låt-ID:t, eller -1. fade är millisekunder: låten tonas in eller tonas ut när n är -1. maskbitar som är inställda är de kanaler som används för låten. 0 betyder alla åtta. De andra kanalerna förblir lediga för sfx.

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

## musicstat

```
musicstat()
```

Tabell över låten som spelas. song och row är låt-ID och raden, eller -1 när ingenting spelas. step och pos har åtta siffror. step[1] är kanal 0, step[8] är kanal 7. step är notindex, eller -1 när den kanalen är tyst. pos är hur långt noten har gått, från 0 till 1. tick är hur många tick den här raden redan har spelats. played är hur många rader som har startats sedan music(), inklusive den som spelas.

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

## songget

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

Mönster-ID lagras i en sångcell. ch är 0..7. 255 betyder att cellen är tom. Detta är live-anropet; kodlänktexten visar fortfarande en äldre argumentordning.

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

## songset

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

Skriver ett mönster-ID i en sångcell. Använd 255 för att rensa cellen.

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

## stat

```
stat(n)
```

Endast ljud. 16..23 är sfx-ID:t på kanal n-16. 24..31 är musikmönstret på kanal n-24. Alla andra n returnerar 0. Bildtiming och RAM är perf och info, inte stat.

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

## Datatabeller

### dget

`dget(t, row, col)`

Läser en cell i datatabellen t från DATA-panelen. Rad och kolumn börjar på 0.

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

### dset

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

Skriver en cell. Använd en sträng för en textkolumn.

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

### dstr

`dstr(t, row, col)`

Cellen som text. Tal konverteras.

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

### drows

`drows(t)`

Hur många rader tabell t har.

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

### dcols

```
dcols(t)
```

Hur många kolumner tabell t har.

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

## Spara och prestationer

### save

```
save(k, v)
```

Lagrar v under nyckel k i sparfilen. v kan vara ett tal, en sträng eller en tabell över dessa. Filen är komprimerad.

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

### load

```
load(k)
```

Läser nyckel k. Returnerar noll när nyckeln saknas.

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

### delsave

```
delsave(k)
```

Tar bort nyckel k från sparfilen.

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

### saveslot

```
saveslot([name])
```

Väljer sparfilen i författarens mapp. Utan namn är spelautomaten speltiteln. Returnerar 1 när filen redan finns.

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

### achieve

```
achieve(i)
```

Låser upp prestation i, från 0 till 15. Lagras lokalt. Ett Steam-bygge sätter också ACH_i. En katalogsession rapporterar det till spelarens konto.

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

### achieved

```
achieved(i)
```

1 när prestation i redan är upplåst.

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

### achdef

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

Skåltext för prestation i, visas i 15 sekunder från skärmens nederkant. spr är en valfri 8×8-platta, texta dess ark.

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

### achclear

```
achclear(i)
```

Rensar prestation i i den lokala filen. En Steam ACH_i som redan var inställd förblir inställd.

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

## Nätverk

### whoami

```
whoami()
```

Ditt spelar-ID, 0..15, medan en session pågår. Offline är det -1.

```
id = whoami()
```

### nhost

```
nhost()
```

1 om den här maskinen är värden, annars 0.

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

### nplayers

```
nplayers()
```

Hur många live-ID:n finns i 0..15.

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

### nlisten

`nlisten()`

Börjar hosta UDP och returnerar LAN-adressen som ip:port.

```
addr = nlisten()
```

### njoin

`njoin(addr)`

Ansluter till ip eller ip:port.

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

### nleave

`nleave()`

Lämnar sessionen. Motparter ser _mp_leave med varför 0.

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

### nsend

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

Skickar ett förlustbringande meddelande. Namnet är händelsen. Upp till fyra extra värden följer. p från 0 till 15 viskningar till den motparten; uteslut p för att skicka till alla.

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

### ncall

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

Samma form som nsend, men meddelandet försöks skickas igen tills det bekräftas. Tre misslyckanden i rad avslutar sessionen.

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

### nlan

`nlan()`

Nuvarande LAN-ip:port efter nlisten.

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

### nstun

`nstun()`

Din publika IP-port från STUN. En gäst på ett annat nätverk kan försöka ansluta till det.

```
pub = nstun()
```

### setturn

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

TURN-reserv när en direkt sökväg misslyckas. setturn() med tomma argument stänger av den. Adressen är host:port.

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

### nlist

`nlist(game)`

Tabell över öppna sessioner för det spelnamnet. Varje post innehåller adress, spelare och namn.

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

### nannounce

`nannounce(game)`

Hjärtslag så den här värden visas i nlist för spelet. Anropa den på en timer medan du är värd.

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

### nkb

`nkb([on])`

Adressfält. nkb(1) öppnar det, nkb(0) döljer det, nkb() talar om om det är öppet. Upp och ner ändrar ett tecken, 0–9 eller en punkt eller ett kolon. A skriver det. X raderar. Tangentbordet skriver inte i det.

```
nkb(1)
```

### nkbstr

`nkbstr([s])`

Text i adressfältet, sedan ändras tecknen uppåt och nedåt. nkbstr("") rensar texten.

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

## _mp

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

Du skriver den här funktionen. Den körs när ett meddelande anländer. from är avsändar-ID. name är händelsen. Upp till fyra värden följer.

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

## _mp_join

`_mp_join(p)`

Du skriver den här funktionen. En peer tog emot id p. Den utlöses också för dig; whoami() är det id:t.

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

## _mp_leave

`_mp_leave(p, why)`

Du skriver den här funktionen. Peer p vänster. Varför 0 är en normal lämna. Varför 1 är en förlorad anslutning.

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

### _mp_update

`_mp_update()`

Du skriver den här funktionen. Den körs var 15:e _update-bildruta, cirka 4 Hz. En bra plats att skicka positioner.

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

### _mp_fail

`_mp_fail()`

Du skriver den här funktionen. Sessionen är död: värden försvann, eller så misslyckades ncall tre gånger.

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

## Coroutines

### cocreate

`cocreate(f)`

Skapar en coroutine från funktion f. Den körs inte förrän coresume.

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

### coresume

`coresume(c, ...)`

Kör coroutine c tills den ger eller returnerar. Vid lyckad: sant och de erhållna värdena. Vid fel: falskt och felet.

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

### costatus

```
costatus(c)
```

"avstängd", "springande" eller "död".

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

### yield

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

Ger kontrollen tillbaka till den som anropade kärnsumman. Värdena blir resultatet av den kärnsumman.

```
yield(1)
```

## Mäta

### info

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

Ritar live-HUD:n i _draw: fps (1 sekunds genomsnitt), upd, draw, px, max, ram, peak, ramc och net när en session är avslutad. info() använder utskriftsmarkören. info(col) anger endast färgen. info(x, y) eller info(x, y, col) placerar den. Siffror använder B, k eller m så att de får plats på en liten skärm. Själva samtalet räknas inte i px.

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

### showstats

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

Samma HUD som info, ritad automatiskt vid 0, 0 efter _draw, ignorerar kamera och klipp. showstats(0) döljer det. showstats(1) eller showstats() visar det.

```
showstats(1)
```

## perf

### perf(n)

Ett HUD-nummer, så att du kan rita det själv. 0 fps, 1 uppdatering, 2 ritningar, 3 px, 4 max (tyngsta bildrutan denna sekund), 5 RAM, 6 RAM-minne (RAM plus den packade spelkassetten), 7 högsta RAM-minne sedan spelkassetten startade.

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