PIX37 Manual

Redigeringsværktøj, derefter kode. Offline-kopi af bogen på pix37.party/docs.

Redaktør

PIX37 DEV er editoren. Du tegner, skriver og lytter til spillet her, og eksporterer derefter en spilkassette eller et separat build. Den kan også afspille en spilkassette, og den kan åbne PIX37.PARTY for at downloade spilkassetter og bruge kataloget. PIX37 DEV EDUCATIONAL er den samme editor, og den er gratis. Den har ingen builds, ingen eksport af spilkassetter og ingen adgang til PIX37.PARTY. PIX37 Console er afspilleren. Den afspiller `.p37` -spilkassetter, og du åbner PIX37.PARTY for at downloade spilkassetter og gennemse kataloget.

Projekter og spilpatroner

En `.pix37` -fil er projektet: kildefaner, sprite-ark, kort, lyd og datatabeller. En `.p37` -fil er den spilkassette, som spilleren kører: tætpakket bytekode.

Gemte filer og præstationer findes i forfattermappen ved siden af appdataene. På Windows ser det sådan ud `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` for en gemt fil, og `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` for præstationer. På Linux er de samme filer `~/.local/share/PIX37/Mira/Pong.p37s` og `~/.local/share/PIX37/Mira/Pong.ach` . Ada er kontoen på computeren. Mira er forfatternavnet, og det navn er mappen. Pong er gem-pladsen, som som standard er spillets titel. `save` skriver `.p37s` . Den første oplåste præstation skriver `.ach` . Hvis filerne allerede er der, læser brugeren dem, når spilkassetten starter.

Udviklermenu

Escape åbner udviklermenuen. Escape, mens et spil kører, vender tilbage til editoren. Bekræft med Enter eller A-knappen (V som standard). Tilbage er B-knappen (C som standard).

Uden indlæsning af noget projekt er menuen: OPRET PROJEKT, INDLÆS PROJEKT, BYG, INDLÆS spilpatron, FORBIND TIL PIX37.PARTY, INDSTILLINGER.

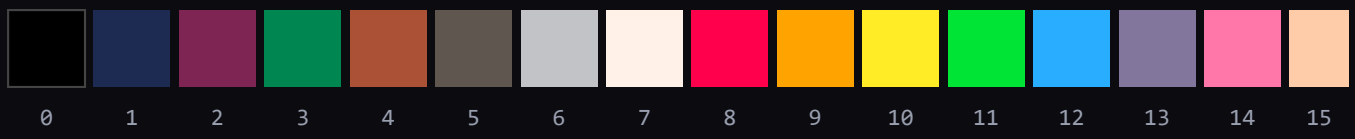
Med et projekt indlæst er det: GEM PROJEKT, INDLÆS PROJEKT, OPRET PROJEKT, AKTIVER, BYG, INDLÆS spilpatron, FORBIND TIL PIX37.PARTY, INDSTILLINGER.

Ctrl+S gemmer det åbne projekt. Ctrl+R kører det.

Opret et projekt

OPRET PROJEKT beder om spilpatronen, før du begynder at lave projektet.

- **VIDEO** — 1-BIT (2 farver), 2-BIT (4), 4-BIT (16), 8-BIT (256).
- **AUDIO** — sampledybde for synthen: 16, 8, 4, 2 eller 1 bit.
- **SKÆRM** — forudindstillinger inkluderer 64×64, 128×128, 160×90, 256×256 og 320×180. Med BRUGERDEFINERET størrelse kan du vælge enhver størrelse fra 1×1 til 512×512.
- **PALETTER** — farvebanker. Den første bank starter som sættet med 16 farver nedenfor. LOAD PALETTE anvender et indbygget sæt: PICO-8 16, PICO-8 32, Game Boy, 1-bit, CGA, Sweetie-16 og en længere liste taget fra Lospec (Resurrect 64, Apollo, Endesga 32 og 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century og andre). Du kan også indlæse en paletfil eller indlæse din egen PNG og tage paletten ud af det billede.



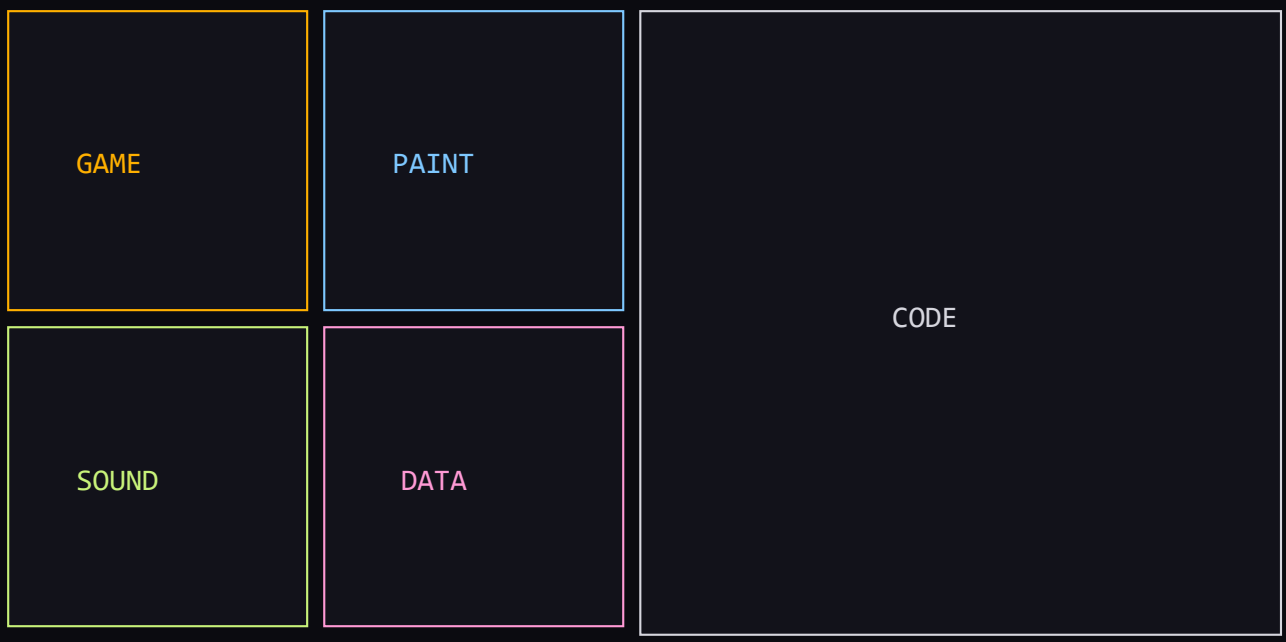
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

lette — black, navy, purple, green, brown, gray, silver, white, red, orange, yellow, lime, blue, indigo, p

Farveindekser brugt af `cls` , `pset` , `spr` og resten. Du kan genmale ethvert indeks i paleteditoren.

De fem paneler

Editoren er GAME, PAINT, CODE, SOUND og DATA, i den rækkefølge. Kortet findes i PAINT. SFX og sange findes i SOUND. Klik på et panel for at arbejde der. Når CODE er i fokus, ignorerer SOUND og DATA bogstavtaster, så indtastning ikke også spiller klaveret.



> Fuldt layout, skrevet `1:1:4:1:1` . Hvert tal angiver, hvor meget af rækken panelet fylder: SPIL, MALING, KODE, LYD, DATA. En lille skærm sidder centreret i en celle på 128×128. En skærm, der er større end 128 px, udvider SPIL-kolonnen.

INDSTILLINGER → GLOBAL → LAYOUT cykler arrangementerne:

Kode	Hvad du ser
`1:1:4:1:1`	Alt. SPIL og LYD stablet, MALING og DATA stablet, KODE til højre.
`1:1:4:1:1 2X`	Samme arrangement. KODE tegnes med dobbelt tæthed, så dobbelt så meget tekst er plads til.
`1:0:1:0:1`	SPIL, DATA, KODE.
`1:0:1:0:1 2X`	Den samme ordning, med den tættere KODE.
`0:1:1:1:1`	Ingen forhåndsvisning af spillet. MALING, KODE, LYD, DATA.
`0:1:1:1:1 2X`	Den samme ordning, med den tættere KODE.
`0:1:4:1:2`	Studio. Bred CODE, en højere DATA-stribe, MALING og LYD.
`0:1:4:1:2 2X`	Den samme ordning, med den tættere KODE.
`0:0:4:0:4`	Kun KODE og DATA, begge store.
`0:0:4:0:4 2X`	Den samme ordning, med den tættere KODE.
`0:1:4:1:0`	MALING, LYD og en bred KODE.
`0:1:4:1:0 2X`	Den samme ordning, med den tættere KODE.
`0:4:0:0:0`	Kun MALING, 256×256, én pixel af arket til én pixel af lærredet.

En `2X` række beholder de samme paneler. CODE-panelet tegnes med dobbelt opløsning og skaleres tilbage til den samme celle, så tekst, faner og margen er halvt så høje, og dobbelt så meget kilde kan passe på skærmen.

SPIL

Forhåndsvisningen af den kørende spilkassette. Klik på SPIL-panelet, eller tryk på Ctrl+R, for at køre forhåndsvisningen. INDSTILLINGER → SPILPANEL er der, hvor panelet konfigureres:

- **Forhåndsvisning** — inaktiv farve, kontinuerlig eller automatisk genindlæsning, når du redigerer.
- **Spil i et nyt vindue** — spillet åbner ved siden af editoren.
- **Skærm** — forhåndsvisning som RGB888 eller RGB565.
- **SPIL 64** — en 64×64 skærm er enten centreret 1:1 i cellen eller skaleret 2× for at fylde den.
- **Shaders** — CRT og filtre i spillet, separat fra filtrene i editoren.
- **Farve** — panelets baggrund.

MALING

Sprite-ark og, på kortsiden af det samme panel, flisekort. Ark kan være større end skærmen. Sprite-forhåndsvisningsstørrelserne er 8, 16, 32, 64 og 128 (128 kun når arket er mindst 128 px). Rul musehjulet hen over sprite-forhåndsvisningen for at zoome ind. Penselstørrelserne er 1, 2, 3 og 4.

Nøgle	Værktøj
B	Blyant
M	Rektangelvalg
Q	Lasso
S	Stempel
G	Spand
jeg	Pipette
V	Flytte
L / Shift+L	Linje / kurve
U / Shift+U	Hult rektangel / hul ellipse
F / Shift+F	Udfyldt rektangel / udfyldt ellipse
D	Dither
[ ]	Mindre/større børste
Ctrl+Z / Ctrl+Y	Fortryd / gentag. Ctrl+Shift+Z gentager også.
Ctrl+C / Ctrl+X / Ctrl+V	Kopier, klip, indsæt
Ctrl+A	Vælg lærredet

Kortet bruger den samme idé, med færre værktøjer: B blyant, M rektangelmarkering, S stempel, V flytning og Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Hold Alt nede for at zoome kortteksturen. Penselstørrelsen på kortet forbliver med maleværktøjerne.

Hver sprite har otte flagbits. Paint sætter dem på spriten; `fget` og `fset` læser dem i spillet.

INDSTILLINGER → MALINGSEDITOR er der, hvor panelet konfigureres:

- **Palet** — hvilken bank der tegnes, plus indlæsning og gemning.
- **Forhåndsvisning** — RGB888 eller RGB565.
- **Penselmarkør** — markøren, mens du maler.
- **Pixel snap** — hvordan penslen snapper til pixels.
- **Stor børste** — en stor børste knipser fra øverste venstre hjørne eller fra hele sit fodaftryk.
- **Løgskaller** — et spøgelse fra andre rammer, og hvilken retning spøgelset vender mod.
- **Kortzoom** — ægte pixels eller fri zoom.
- **Farver** — farverne i malingsgrænsefladen.

KODE

Faner er separate filer i projektet. Klik på et fanebladnavn for at omdøbe det; Enter bekræfter. Flere faner, der hver især definerer `_init` , `_update` eller `_draw` er sat sammen i tabulatorrækkefølge: den første faneblad kører først, den sidste faneblad kører sidst. Kode, der ikke er inde i en hook, bliver en del af `_init` . En spilkassette uden hooks overhovedet behandles som én `_draw` .

Nøgle	Handling
Pile	Flyt cirkumfleksknappen. Shift udvider markeringen.
Ctrl+Venstre / Højre	Med ord.
Hjem / Slut, Ctrl+Hjem / Slut	Start eller slutning af linjen eller af tabulatoren.
Faneblad	Indsæt to mellemrum. Accepter navnet i fuldførelseslisten.
Indtast	Ny linje, indrykket som den forrige. Accepterer en fuldførelse.
Ctrl+C / X / V	Kopier, klip, indsæt.
Ctrl+Z / Ctrl+Y	Fortryd / gentag.
Ctrl+A	Vælg fanen.
Ctrl+D	Dupliker linjen eller markeringen nedad.
Ctrl+/	Kommentér eller fjern kommentar med `--` .

Ctrl+F	Find. F3 hopper til næste hit. Escape lukker baren.
Ctrl+G	Gå til et linjenummer.
Flugt	Lukker først find, gå til eller hintlisten. En anden Escape forlader editoren og går videre til udviklermenuen.

Der dukker hints op, mens du skriver et funktionsnavn. Enter eller Tab indsætter navnet, og et `(` hvis FÆRDIG (er aktiveret i indstillingerne. INDSTILLINGER → KODEDITOR:

- **Søg** — Ctrl+F i den aktive fane eller på tværs af alle faner.
- **Tekst høj** — beskær glyffer til 5 px, eller vis den fulde glyff.
- **Strengstørrelse** — skriftstørrelsen for strengliteraler.
- **Kommentarskrifttype** — 3×5 eller Fusion-skrifttypen i det aktuelle brugergrænsefladesprog.
- **Fuldført (** — om fuldførelsen indtaster parentesen.
- **Kodetips** — til eller fra.
- **Tipssprog** — menusproget eller engelsk.
- **Cirkumfleks** — ekstra forskydning på tomme linjer.
- **Innermarg** — linjenumre.
- **Farve** — baggrund, kommentarer, tabulatorer, tal, operatorer, nøgleord, strenge, programfunktioner, dine funktioner, variabler, parenteser, cirkumfleks, markering, fejllinje, advarsel.

SUND

Panelet har to faner. SFX er ét mønster: en kort række toner. SOUND er en sang: den placerer disse mønstre på otte kanaler. TIC er, hvor mange ticks ét trin venter. Uret er 120 ticks i sekundet, og ét trin er en sekstende node, så 15 ticks er 120 slag i minuttet. En højere TIC er et langsommere mønster. To mønstre forbliver i takt, når den ene TIC deler den anden: 15 og 30 gør, 15 og 8 gør ikke.

Klaveret spiller, når en node vælges med musen. Hvide tangenter er ZXCVBNM (CDEFGAB). Sorte tangenter er SDGHJ (C# D# F# G# A#). Hold musen over en vilkårlig værdi, og skriv et tal, eller hold musen over den, og rul med hjulet. Tonehøjdetal i kode går fra 0 (C-0) til 63 (D#5).

På ét mønster er LOOP og LEN de samme to tal. OUT over 0 looper fra IN til OUT, og titlen lyder LOOP. OUT ved 0 og IN over 0 er LEN: så mange toner spilles fra starten, så slutter mønsteret, og titlen lyder LEN. Begge ved 0 spiller hvert trin.

Hvor meget af synthen du kan redigere afhænger af den valgte bitdybde for spillekassetten. INDSTILLINGER → SFX EDITOR angiver dybden, om en tone automatisk går frem (fra, tonekæde eller feltkæde), om Space genstarter afspilningen eller slår start og stop til/fra, samt editorens farver.

16-bit

Hvert trin har en tone, en lydstyrke, en bølge og en effekt. Lydstyrken er fra 0 til 7. 0 er lydløs.

Bølger:

- **0 trekant**
- **1 vippet sav**
- **2 sav**
- **3 kvadrat**
- **4 pulser** — en smal puls
- **5 orgel**
- **6 støj**
- **7-faset**
- **8 PWM** — en firkant, hvis bredde bevæger sig
- **9 pulser 25%**
- **10 sinus**
- **11 metallisk støj**
- **12 FM-klokke**
- **13 pluk**
- **14 supersav**
- **15 PWM (C64)**

Effekter på en enkelt tone:

- **0 ingen**
- **1 slide** — tonehøjde og lydstyrke glider til den næste tone på tværs af dette trin
- **2 vibrato** — tonehøjde-svingninger
- **3 drop** — tonehøjden falder til stilhed, det sædvanlige spark
- **4 fade ind**
- **5 fade ud**
- **6 arpeggioer** — går denne tone og de næste tre, én gang for hver 8. ticks. Ved TIC 8 eller derunder, hver 4. ticks.
- **7 hurtige arpeggioer** — de samme fire toner, hver 4. tick, eller hver 2. når TIC er 8 eller derunder.

Mønsteroverskriften, over noterne:

- **TIC** — tick per step, fra 1 til 255. 15 er 120 slag i minuttet. Skriv tallet, eller skift det et trin ad gangen.
- **LOOP IND / UD** — med OUT over 0, looper mønsteret fra IN til OUT. Hvis OUT ikke står på 0, bliver titlen LEN: IN er, hvor mange toner der spilles, og 0 betyder hele mønsteret.
- **STØJ** — hele mønsteret bliver til støj.
- **CRUSH** — en livlig variant af hver bølge.
- **DTUNE** — 0 slukket, 1 et let omkvæd, 2 et stærkt omkvæd.
- **RVERB** — rumklang: 0 slukket, 1 kort, 2 lang.
- **DAMPN** — et lavpasfilter: 0 slukket, 1 lys, 2 stærk.

- **LEVEL** — lydstyrke for hele mønsteret, 0 til 7.
- **SVING** — blande sig. Ulige trin venter lidt længere, lige trin lidt kortere.
- **TRIN** — hvor mange trin mønsteret bruger: 32, 24, 16 eller 12.

8-bit

Et trin har en node og en bølge. De fire bølger er firkant, trekant, sav og støj. Headeren har TIC, LOOP IN/OUT, FX og LEVEL. TIC er en af 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60. FX er én effekt for hele mønsteret: 0 ingen, 1 vibrato, 2 arpeggio, 3 fade out. LEVEL er 0 til 3.

4-bit

Et trin er kun en node. Overskriften har TIC, LOOP (tændt eller slukket), WAVE (de samme fire bølger) og OCT (hvilken oktav noderne sidder i). TIC er en af 4, 6, 8, 10, 12, 15, 20, 30.

2-bit

Et trin er tomt, en tromme, et hit eller et crush. Hovedet har TIC og BASE. TIC er 8, 15, 16 eller 30. BASE angiver, hvor højt trommen og hiten sidder.

1-bit

Et trin er en note. TIC er 8, 15, 16 eller 30, de samme fire hastigheder som 2-bit.

Sange

Fanen LYD er den åbne sang. Otte kolonner er kanalerne. En celle er et mønster-id. 255, tegnet tom, betyder, at kanalen er lydløs på den række. X-kolonnen er en kommando: fortsæt, stop, loopstart eller loopslut. Stop afslutter sangen efter rækken. Loopslut hopper tilbage til den sidste loopstart.

En række slutter, når mønsteret yderst til venstre med LOOP OUT som 0 slutter. Mønstre længere til højre gør ikke rækken længere: et længere mønster klippes, et kortere mønster starter forfra ved sit første trin, indtil rækken slutter. Et mønster med LOOP OUT over 0 løber i en løkke mellem LOOP IND og LOOP OUT og afslutter ikke rækken. Hvis alle mønstre på rækken løber i en løkke, forbliver rækken der.

Genveje

Nøgle	Handling
Klavernøgler	Skriv en note om det trin, der er valgt med musen.
Op / Ned	Ændr værdien under markøren.

Plads	Afspil den åbne SFX eller den åbne sang fra første række. Start/stop eller genstart fra SFX EDITOR-indstillingerne.
Ctrl+Z / Ctrl+Y	Fortryd / gentag.
Ctrl+C / Ctrl+X / Ctrl+V	Kopier, klip, indsæt det markerede.

DATA

Navngivne tabeller. Hver tabel består af rækker og kolonner. Fra koden bruger `dget` , `dset` , `dstr` , `drows` og `dcols` disse tabeller. Række- og kolonnenumre i koden starter ved 0.

Kolonnetyper, cyklisk i overskriften:

- **bool** — en bit.
- **2** — to bits.
- **4** — fire bits.
- **8** — en byte.
- **16** — 16 bit, for id'er og større tal.
- **spr** — et sprite-id.
- **str** — tekst, tegnet med systemskrifttypen.
- **str8** og **str10** — den samme tekst, tegnet højere.
- **f16** og **f32** — en halvpræcisionsflyder og en fuldflyder.

Når markøren er på en kolonneoverskrift, ændrer Ctrl+rul kolonnens type. En almindelig rulning bevæger sig på tværs af kolonner eller ned ad rækker, alt efter hvad INDSTILLINGER → DATA EDITOR vælger. Ctrl skifter den retning, hvilket er grunden til, at Ctrl+rul på en overskrift ændrer typen, når hjulet er indstillet til kolonner.

Piletasterne bevæger sig. Enter redigerer en celle. INDSTILLINGER → DATA EDITOR:

- **Hjul** — rul i kolonner eller rul i rækker.
- **Farver** — gitteret, overskriften og markeringen.

Aktiver

ASSETS importerer og eksporterer det åbne element, ikke hele projektet. Sprites bruger den åbne sprite i PAINT på det åbne ark. Kortet er det åbne kort. SFX er det åbne mønster. Musik er den åbne sang. Data er den åbne tabel. Kode er undtagelsen: hver tabulator skrives, og hver tabulator læses tilbage.

Slags	Fil	Hvad bevæger sig
Sprites	`.png`	Den åbne sprite.

Kort	`.p37map`	Det åbne kort.
SFX	`.p37sfx`	Det åbne mønster.
Musik	`.p37mus`	Den åbne sang. Mønsternumrene forbliver som de er. Eksporter SFX separat, hvis du har brug for disse lyde.
Data	`.p37dat`	Det åbne bord.
Kode	`.lua`	Hver fane.

Indstillinger

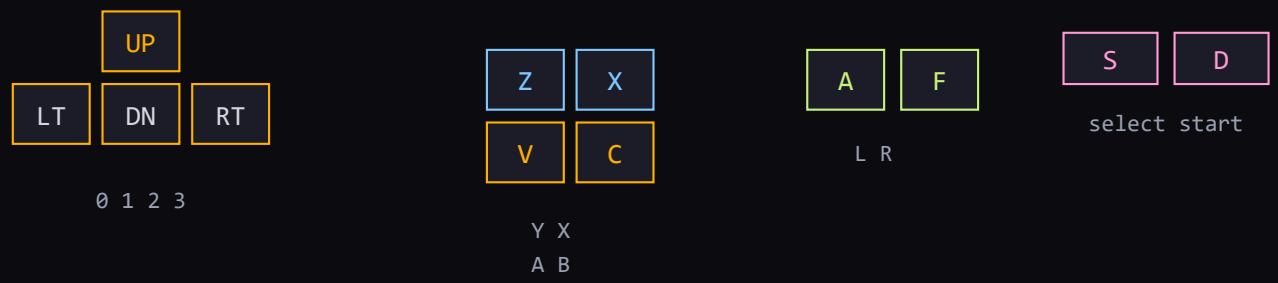
Rodlisten:

Side	Hvad den gemmer
PROFILER	Gem og indlæs alle indstillinger. Ny, omdøb, overskriv, slet. Kopiér eller indsæt profilen som tekst.
Menu	Vindue, taster, menufarver, valgfri baggrund.
SPILPANEL	Forhåndsvisning og afspilningsvinduet.
KODEDITOR	Fremhæv og hints.
MALINGSREDAKTOR	Pensel, palet, kortvisning.
SFX-REDAKTOR	Synthesizer og nodeindtastning.
DATAREDAKTØR	Hjulretning og farver.
SPROG	Menu- og tipsprog.
LICENSER	Licenstekster, inklusive Fusion Pixel-skrifttypen og SDL3.

GLOBAL:

- **LAYOUT** — panelarrangement, ovenfor.
- **LYDSTYRKE** — masterniveau, 0 til 10.
- **TASTBINDINGER** — taster til hver af de 8 spillere. Tryk på en tast for at fange den. Vejledning, ABXYLR, vælg.
- **Præsenter** — spring inaktive billeder over, eller tegn alle billeder.
- **Fuldskærm** — uden rammer.
- **Opdater** — vindues-FPS. Venstre og højre trin 60, 90, 120 og maks. Indtastning indtaster et tal op til 1000. 0 betyder ingen grænse.
- **Vsync** — vent på visningen mellem billederne.

- **Skala** — heltalsvinduesmultiplikator.
- **Shaders** — CRT og filtre i editoren, ikke i spillet.
- **Farve** — menutekst og fremhævning.
- **Baggrund** — en valgfri PNG, beskåret til 128, bag menuen. Ryd fjerner den.
- **Editor-markør** — markørform.
- **Mus** — flydende markør eller fastgjort til pixels.
- **UI-hastighed** — hvor hurtigt en holdt retning gentages gennem en liste.
- **Stifinder** — fillister starter ved computerens rod eller i exe-mappen.



Standardtastatur. A er bekræftelse i menuer. B er tilbage. I et spil er disse knapper 4 og 5. Vælg (S) er knap 10, ikke en menugenvej.

Knap	Rolle	Standardnøgle
0	Venstre	Venstre pil
1	Højre	Højre pil
2	Op	Pil op
3	Ned	Pil ned
4	A, bekræft	V
5	B, ryg	C
6	X	X
7	Y	Z
8	R	F
9	L	EN
10	Vælge	S
11	Starte	D

Byg

Vælg en platform: Windows, Linux, Mac, Android, iOS eller Web.

- **Navn** — filnavnet på buildet.
- **Forfatter** — på Windows og Linux. Dette er mappenavnet for gemte filer.
- **Steam-app-id** — på Windows og Linux. Tom betyder intet Steamworks. Et udfyldt id knytter `achieve(0)` … `achieve(15)` til Steam-achievements ACH_0…ACH_15.
- **Id** — bundt-id, hvor platformen beder om et.
- **Skjul kode** — på desktop- og mobiloverlejringer. Det forsinker `peek` af spilkassetten. Det er ikke en lås.
- **Fuldskærm** — den eksporterede afspiller starter i fuldskærmsvisning.
- **Shaders** — op til tre lag indbygget i afspilleren.
- **Download .p37** — Kun web. Viser en download-post i pausemenuen.
- **Ikon** — en flise fra et sprite-ark: ark, størrelse, x, y. Eller et PNG-omslag, du beskærer.
- **Eksporter** — vælg mappen. Menuen viser enten BYGNING GEMMT eller BYGNING MISLYKKET.

Lav et build med internettet tændt. Splash'en kan anonymt spørge serveren, om dette builds hash er kendt. Denne kontrol er der af to grunde. Den beskytter mig, forfatteren af PIX37, mod kopier af værktøjet, som ikke var mine at give væk. Den beskytter også den person, der lavede spillet. At tage en andens spil og liste det som dit eget er ved at blive almindeligt på telefonmarkedet, såsom Play (Android) eller iOS. Spillet starter stadig. Hvis maskinen er offline, siger introen, at spillet ikke er forbundet. Den linje er også et hint til spilleren om, at kopien muligvis ikke kommer fra en lovlig kilde. Det er en lille kontrol, så hvis dit spil blev stjålet og solgt uden dit samtykke af en anden, kan du fremsætte den påstand i en butik såsom Play (Android), Steam eller itch.io. Serveren gemmer det tidspunkt, hvor hashen først blev registreret, ved siden af hashen, og ændrer det ikke senere. Det tidligere tidspunkt er det originale build. En hash registreret efter det er en senere kopi.

Jeg ønsker, at denne kontrol skal beskytte de folk, der laver spil. Spillemotoren gør allerede en stor del af arbejdet for dem, og det er en af de ting. Jeg forstår, at det kan være kontroversielt. Hvis fællesskabet beslutter, at det er bedre at droppe denne mikro-DRM, vil jeg gøre det.

En gammel spilpatron fra en tidligere editor kan nægte at boote efter sprogændringer. Åbn `.pix37` -projektet i denne editor og eksporter en ny `.p37` .

Opret forbindelse til PIX37.PARTY

FORBINDELSE TIL PIX37.PARTY er inde i appen. Der er ingen loginside i browseren. Du logger ind med en Steam-sessionsbillet eller med itch.io ved hjælp af tastaturet og musen. Tjenesten viser et unikt visningsnavn på 2 til 16 tegn. Licensnøglen og Steam-id'et vises ikke til andre spillere.

Efter servicevilkårene er hubben Favoritter, Katalog, Rangering, Spillere og din profil.

En udviklerlicens er redaktøren. Du laver spillet, du uploader det, og du efterlader stjerner. En spillerlicens er spilleren. Hjerter er fra folk, der spiller på en runner.

- **Katalog** — gennemse spil. Filtre dækker genre (op til tre), skærmstørrelse, spilkassettestørrelse og om uploadet inkluderer et `.pix37` projekt.
- **Favoritter** — spil du har markeret.
- **Upload** — kræver en udviklerlicens. Du sender en `.p37`, og du kan vedhæfte `.pix37`-projektet. En `.p37` er pakket bytecode. Den er svær at læse, og den kan blive tilsløret med vilje. En `.pix37` er den projektfil, du rent faktisk arbejder i: kilden, billederne og lyden, i en form, som en anden person kan åbne og lære af. Ved at sende den kan andre skabere studere spillet i stedet for en komprimeret spilkassette. Titel, forfatter, genrer, cover, ikon og præstationsnavne følger med. RAM-nummeret, der sendes op, er det højdepunkt, som spilkassetten nåede.
- **Kommentarer og præstationer** — på spilsiden. Oplåsning af en præstation, mens du er logget ind, gemmes på den konto såvel som på maskinen.
- **Rangering** — udviklere, sorteret efter stjerner eller hjerter. Hver række viser begge tællinger. Åbn et navn for at se den pågældende forfatters spil, sorteret efter hjerter.
- **Spillere** — konti sorteret efter oplåste præstationer.

Kode

Sproget er et lille script i Lua-stil. Denne halvdel starter fra et tomt projekt, lærer sproget og viser derefter et kort eksempel på hver funktion.

Et første program

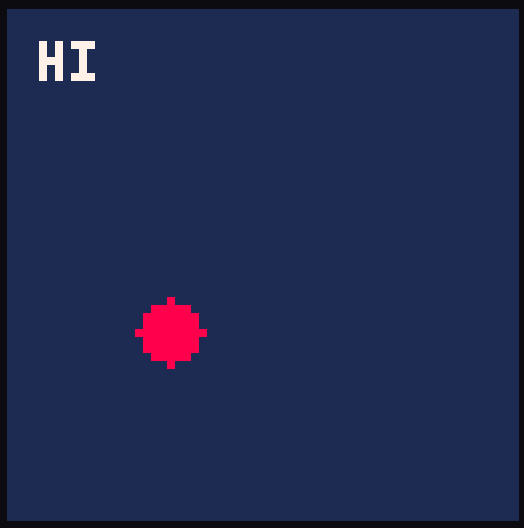
Opret et projekt i 64×64, 4-bit farve. Åbn CODE. Erstat fanen med:

```
_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 kører den. Piletasterne flytter cirklen. Escape stopper spillet og vender tilbage til editoren.



De samme tre kald: `cls(1)` maler den marineblå skærm, `print` tegner HI, og `circfill` tegner den lyserøde cirkel.

Hvad kører, og hvornår

- `_init` kører én gang.
- `_update` kører ved 60 Hz. Sæt bevægelse og knapper her.
- `_draw` maler. Uden et fps-argument maler den så hurtigt, som computeren og skærmen kan. `_draw(30)` begrænser malingen til 30 fps og lader logikken være på 60.
- `_set_fps(n)` ændrer den grænse senere. 0 fjerner den.

Navne skelner ikke mellem store og små bogstaver; editoren gemmer dem med små bogstaver. `function` og `fun` er det samme nøgleord.

Sprog

Grundlæggende programmering

Et program er en liste over linjer, som spillekassetten kører. Hver overskrift viser først linjens form og derefter et lille eksempel. En linje, der starter med `--` er en kommentar: en note til dig. Spillet springer den over.

Variabler

En variabel er et navn, der indeholder en værdi. Du vælger navnet. Senere linjer kan læse det, og de kan gemme en ny værdi med samme navn.

Formen er `name = value` . `local name = value` holder navnet inde i den aktuelle blok, så resten af spillekassetten ikke kan se det. Uden `local` er navnet globalt, og alle faner 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, at der ikke er nogen værdi. `true` og `false` er ja og nej. Et navn, du aldrig har angivet, læses som `nil` .

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

Tabeller

En tabel indeholder mange værdier under ét navn. En liste bruger tal. Det første element er 1, ikke 0. En post bruger navne. En tabel kan indeholde en anden tabel.

Listeform: `{ value, value }` . Postform: `{ name = value, name = value }` . En liste over poster er en tabel, hvis elementer er 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
```

Tilstand og krop

En betingelse er et spørgsmål med et ja- eller nej-svar, såsom `hp > 0` eller `btnd(4)` . En krop er den gruppe af linjer, der løber, når du går ind i en blok. `if` og every loop bruger den samme form: en betingelse, derefter en krop, derefter `end` . `repeat` afsluttes med `until` i stedet for `end` .

Hvis

`if` kører kun sin krop, når betingelsen er sand. `elseif` forsøger en anden betingelse. `else` kører, når alle betingelser er falske.

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

Løkker

En løkke gentager en krop. `while` kontrollerer først betingelsen og fortsætter, mens den forbliver sand. `repeat` kører kroppen og kontrollerer derefter, så kroppen altid kører mindst én gang. `for` counts`. Det sidste tal er inkluderet. Uden et tredje tal er trinnet 1, så hvis starten allerede er forbi slutningen, kører kroppen ikke. Nedtælling kræver et negativt trin.

```
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 rundt ved et bord

`all` besøger hvert element på en liste, fra det første. `pairs` besøger hvert navn og den værdi, der er gemt under det navn.

```
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 er en navngivet opskrift. Formen er `function name(arguments)` , derefter en krop og derefter `end` . Du kalder den ved at skrive navnet og parenteser. `return` sender en værdi tilbage, og den kan sende mere end én. Argumenterne er lokale værdier for den pågældende funktion.

```
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` er en funktion, der kan se de lokale omkring sig. Det er en lukning. En tabel kan indeholde én, og du kalder den med en prik.

```
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, og at vælge

vender `not` en betingelse. `and` er kun sand, når begge sider er sande. `or` er sand, når begge sider er sande. `a and b or c` vælger `b` når `a` er sand, ellers `c` . `cond ? a : b` er den samme idé på én linje: `a` når betingelsen er sand, `b` når den ikke er 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 og tekst

`+ - * /` er addering, subtraktion, gange og dividering. `//` dividerer og beholder det hele tal. `%` er resten. `^` er potens. `..` limer to tekster sammen. `#t` er antallet af elementer på en liste. Sammenligninger er `==` (samme), `~=` eller `!=` (forskellige), og `< > <= >=` . `x += 1` lægger 1 til x. Den samme form fungerer for `-= *= /= ^= %=` .

```
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
```

Hvor en lokal bor

`local` varer indtil den `end` , der lukker dens blok. `if` / `end` er dens egen blok. Du kan nævne en local over den linje, der sætter den, men indtil den linje kører, er værdien `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
```

Hopping

`::label::` markerer en linje. `goto label` hopper dertil, men kun inden for den samme funktion.

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

Hukommelse, kort sagt

`peek` and `poke` -kig på levende pixels gennem én 16-bit adresse. De gemmer ikke en anden kopi af spillekassetten. Adresserne `0x5F00` til `0x5FFF` er registre: kamera, klip (skrivebeskyttet), tegnefarve, fyldmønster, de tre palet-omlægninger og sidetallene. Bytes på begge sider af det hul, 65280 af dem, er pixelvinduet.

Når skærmen passer ind i de 65280 bytes, 64×64 og 128×128 gør, og 256×256 ikke gør, starter skærmen ved adresse 0, række for række. Det, der er tilbage, er sprite-arket valgt af `0x5F41` (0 er ark 0). Hvis arket er større end den resterende plads, vælger `0x5F42` siden, og kortet er ikke i vinduet. Hvis arket passer, er halen kortceller fra kortet valgt af `0x5F43` , en eller to bytes hver, og `0x5F44` sider kortet, når det ikke passer ind i halen.

Når skærmen ikke passer, er hele vinduet én side på 65280 pixels. `poke(0x5F40, n)` vender siden på skærmen. `poke(0x5F41, n)` med n andet end 0 viser ark n i stedet, og ark 0 har ingen visning i denne tilstand. `poke(0x5F43, n)` med n andet end 0 viser kort n, og det vinder over arket. Kort 0 i denne tilstand er kun `mget` og `mset` .

Et `peek` , der ikke lander på en pixel eller et register, returnerer 0. Et `poke` der gør ingenting. Sprite-flag, SFX og DATA-tabeller er ikke i dette vindue. Brug `fget` , `sfxget` og `dget` .

Praktisk brug

Hver funktion er angivet nederst på denne side med praktiske anvendelser af PIX37-funktioner. Linkene under hvert eksempel åbner mere om funktionerne.



Jævn bevægelse

En 128×128 spillekassette, set ovenfra. Sprite 1 er allerede tegnet. Pile accelererer i alle retninger, og friktion bremser dig. Spriten vikler sig: en halvdel, der forlader bunden, tegnes øverst, og det samme gælder for de andre kanter.

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



Tjekker om det næste træk er blokeret

En 64×64 spilpatron. Gulvet og den højre væg er allerede på kortet, malet med sprite 3, og flag 0 er sat på den sprite. Pile bevæger sprite 1. Ordet øverst siger blokeret, når det næste trin ville overlappe en solid flise, og fri, når det ikke ville.

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



En klud med flip

Tryk på A. Visningen tegnes med flip(), én sort række ad gangen, i stedet for indeni _draw(). _draw forbliver ude af vejen, mens det kører, så hver række forbliver på skærmen.

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



En tabel med funktioner (klasse)

Sprite 1 er allerede tegnet. make_monster fungerer som en klasse. Den returnerer en tabel, og update og draw er metoder, der beholder monsterets egne x og y. Tallet øverst angiver, hvor langt det har gået.

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



Fire måder at læse en knap på

A er knap 4. Hold den nede, og opladningen stiger. Slip den, og opladningen går tilbage til 0. Op og ned gentages, mens den holdes nede, så menulinjen fortsætter med at bevæge sig. btnd er den første frame af et tryk, som denne spillekassette kun bruger som starten på det hold.

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



Skyd med musen

Sprite 1 er allerede tegnet i midten. En hvid prik er markøren. Venstreklik skyder en pixel mod den. Hjulet ændrer hastigheden, trykt i hjørnet.

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

 Skriv et navn og bekræft det

Tryk på A for at åbne feltet. Op og ned ændrer et blinkende bogstav. A skriver det, X sletter, Y er mellemrum, og B bekræfter. Linjen siger derefter dit navn.

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

 Kamera på afspilleren

En 128×128 spilkassette. Sprite 1 starter midt på et gulv af fliser. Pilene bevæger sig. Kameraet holder spriten centreret. x og y i hjørnet ruller ikke, fordi camera() uden argumenter sætter visningen tilbage for HUD'en.

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

 Et par linjer fra en visuel roman

Linjerne er i DATA, på engelsk og polsk. A fortsætter, eller vælger forbliver, når et valg vises på skærmen. B vælger at forlade et valg, og ellers bytter sprog. X bytter sprog, mens et valg vises. Dialogen bruger skrifttypen 8px, så polske bogstaver tegnes.

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

 Mal med kig og poke

En 64×64 spilkassette, så adresse 0 er den øverste venstre pixel, og den næste række starter ved 64. Spilkassetten skriver pixels, kopierer en række, rydder en række og udskriver de værdier, den læste tilbage.

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

 En liste du kan tilføje til

A lægger et tal sammen, op til 8. B fjerner det sidste. X fjerner det første. Listen udskrives ned langs skærmen.

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

 Et navn taget fra hinanden

Spilpatronen deler ada,12,blå og skriver navnet, dets første bogstav, 12, og ordet blå.

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

 Spil en node, og ændr den derefter

Mønster 0 er én node langt, og hver gentagelse har den samme lydstyrke. A spiller det igen, når den node er færdig. B hæver tonehøjden og spiller den node.

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

 En sang skrevet ind i spillekassetten

To mønstre looper på række 0, et på hver af de første to kanaler. Rækken forbliver 0. ch0 og ch1 er den tone, hver kanal er på. Redigering af mønsteret, LEN og LOOP i spillekassetten er en måde at se, hvad disse kontroller gør i musikpanelet.

`songget` `songset` `music` `musicstat` `cls` `print` `_init` `_draw`

 Tre monstre i DATA

DATA-panelet har allerede en navnekolonne og en hp-kolonne med slim, flagermus og øje. A gør slimet ondt. Tallene til højre er dens hp.

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

 Husk scoren

A tilføjer et point og gemmer det. Afslut og åbn spillekassetten igen: nummeret er der stadig. B glemmer det.

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

  Et andet kapitel i en anden fil

To spilkassetter deler gemte spilleautomat-missionen. Del 1: A markerer spillet som færdigt. Del 2, der åbnes først, siger, at del 1 skal afsluttes. Derefter åbnes del 2, og B sletter gemningen, så del 1 kan spilles igen.

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

 Én præstation

Sprite 8 er allerede trukket. Tryk ti gange på A. Toasten navngiver præstationen, og sprite 8 forbliver, fordi den er låst op. B rydder præstationen i den lokale fil og nulstiller tælleren.

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

 Et rum man kan gå ind i

En 256×256 spilpatron. En hvid prik er markøren. Menuen er vært, tilmeld eller serverlisten. Værten åbner et rum med navnet gå og starter spillet. Tilmeld ruller gennem en adresse (cifre, en prik, et kolon), og B forbinder. Serverlisten bevæger sig op og ned, og A slutter sig til rækken. I spillet går pilene sprite 1. En mønt deles: værten flytter den, og når den samles op, lægges 1 til spillerens score og den næste gyder. Hver sprite, inklusive din, viser den score ved siden af. Hjørnet viser, hvor mange spillere der er i, LAN-adressen og

værtens offentlige adresse. B forlader: du og de andre vender tilbage til menuen, og den næste vært starter et nyt rum ved score 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 der venter

A viser den næste linje. Funktionen holder pause ved hvert udbytte, indtil du trykker på A igen.

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

 Vent et halvt sekund

Ordet ændrer sig fra ... til hej efter cirka et halvt sekund. Skærmen bliver ved med at tegne, mens korrutinen er sat på pause.

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

 Cirkler, der ankommer en efter en

Fem cirkler drypper ind fra toppen, to billeder fra hinanden, og falder derefter.

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

 To billeder fra uret

Sprite 1 og sprite 2 er to billeder af den samme figur. Billedet bytter om fire gange i sekundet.

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

Hver funktion

Argumenter i firkantede parenteser er valgfrie. Indekser for tabeller og strenge starter ved 1, undtagen DATA-rækker og -kolonner, som starter ved 0. Farver er paletindekser.

Indeks

`_tegne`
`_init`
`_mp`
`_mp_fejl`
`_mp_tilslut`
`_mp_leave`
`_mp_opdatering`
`_set_fps`
`_opdatering`
`abs`
`achclear`
`achdef`
`opnå`
`opnået`
`tilføje`
`alle`
`hævde`
`atan2`

band
bnot
bor
btn
btnd
btnp
btnu
bxor
kamera
loft
chr
cirk
cirkulere
klikl
klikm
clickr
klip
cls
samskabe
farve
corsume
cos
costatus
tælle
markøren
dcols
del
delikatesseforretning
delsave
dget
drukner
dset
dstr
fget
fillp
vende
flr
foreach
fset
info
ipairs
tast af
tast
tastestat
keystr
linje
belastning
log
lshr
kort
max
memcpy
memset
mget
midt
min
mus
mset
musik
musikstat
meddele
ncall
næste
nhost
njoin
nkb
nkbstr
nlan
nleave
nliste
nlyt
bemærk
nspillere
nsend
nstun
ord
oval
ovalfyld
pakke
par
kammerat
palrgb
palt
pcall
kig
kig 2
kig4
perf
pget
stikke
stikke 2
stikke 4
print
print10
print8
pset
rekt
rectfill
rnd
rotl
rotr
ret
udfylde
gem
spareslot
skærmen
rulle
vælge
sætkort
sætpalet
setspr
settex
omdrejning
sfx
sfxget
sfxhdr
sfxset
sfxstat
sget
sgn
shl
showstats
shr

synd	srand	tline
sangget	sset	tonum
sangsæt	sspr	tostr
split	stat	pakke ud
spr	sub	whoami
sqrt	tid	udbytte

Livscyklus

_init

`_init() ... end`

Kører én gang, når spilkassetten starter. Kode, der sidder uden for en hvilken som helst hook, foldes også ind i _init.

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

_update

`_update() ... end`

Spillogik. Kaldes ved 60 Hz, selv når _draw er begrænset, eller værten er langsommere. Læs knapper her.

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

_draw

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

Tegner én frame. Uden argument, eller 0, kører den så ofte som værten præsenterer. Send n (1..999) for at begrænse tegningen til n frames per sekund. _update forbliver ved 60 Hz.

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

_set_fps

`_set_fps(n)`

Ændrer _draw-grænsen, mens spillet kører. 0 betyder ikke-grænse. n er begrænset til 0..999.

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

flip

`flip()`

Præsenterer skærmen og giver efter indtil næste frame. Brug den kun, hvis du tegner din egen frame-loop. Vent ikke med at holde dig oppe efter den.

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

time

`time()`

Sekunder siden _init.

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

Input

btn

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

1 mens knap b holdes nede, ellers 0. p er spilleren, 0..7. Uden argumenter returnerer den en bitmaske for spiller 0. Knapper: 0 venstre, 1 højre, 2 op, 3 ned, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 vælg, 11 start.

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

btnd

`btnd(b, [p])`

Kun sand på _update-rammen, når knap b trykkes ned.

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

btnp

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

Sand ved tryk, og derefter igen mens den holdes nede: 30 tick forsinkelse, derefter hvert 8. tick. Uden en knap returneres en maske.

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

btnu

`btnu(b, [p])`

Kun sandt på rammen, hvor knappen slippes.

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

mouse

`mouse()`

Returnerer musens x, y i spillets pixels.

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

clickl

`clickl([p])`

Venstre museknap. clickl() holdes nede. clickl(1) er pressekanten.

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

clickr

`clickr([p])`

Højre museknap. Samme kant med 1 argument som ved klik.

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

clickm

`clickm([p])`

Midterste knap eller et klik med scrollhjulet.

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

scroll

`scroll()`

Musehjulet bevæger sig i denne ramme. Positivt og negativt afhænger af hjulets retning.

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

keyon

`keyon()`

Åbner et tekstfelt, der styres af knapper. Op og ned ændrer et bogstav fra A til Z. A skriver det bogstav. X sletter. Y er mellemrum. B bekræfter og lukker feltet. Tastaturet skriver ikke i det.

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

keyoff

`keyoff()`

Skjuler tekstfeltet.

```
keyoff()
```

keystr

`keystr([s])`

Tekst i feltet, derefter 1 på rammen B bekræfter, og derefter ændres bogstavet op og ned. Bogstavet er tomt, når feltet lukkes. keystr("") sletter teksten.

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

keystat

`keystat()`

1 mens tekstfeltet er åbent, ellers 0.

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

Tegning

screen

`screen()`

Aktuel skærmstørrelse for spilkassetten i pixels. Returnerer bredde og derefter højde. Dette er spillets opløsning, ikke editorvinduet.

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

cls

`cls([col])`

Rydder skærmen. Standardfarven er 0.

```
cls(1)
```

color

`color(col)`

Angiver den tegnefarve, der bruges, når et senere kald udelader en farve. Uden et argument returneres den aktuelle farve.

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

camera

`camera([x, y])`

Offseter hvert draw-kald. camera() nulstilles til 0, 0.

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

clip

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

Begrænser tegningen til et rektangel. Et femte argument forskelligt fra nul skærer det aktuelle klip. clip() gendanner fuldskærmsvisningen. Peek af klipregistrene er skrivebeskyttet.

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

cursor

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

Angiver, hvor den næste udskrift uden koordinater skal være, og valgfrit returnerer color.cursor() x, y.

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

pset

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

Angiver én skærmpixel. Tæller med i pixelstatistikken.

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

pget

`pget(x, y)`

Farven på skærmpixelen ved x, y.

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

print

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

Tegner 3×5 tekst (UTF-8, beskåret fra Fusion-skrifttypen). Uden x og y udskrives der ved markøren og flyttes til næste linje.

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

log

`log(...)`

Tilføjer argumenterne, adskilt af mellemrum, til en tekstbuffer. pix37_run udskriver den buffer ved stdout. Den tegner ikke.

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

print8

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

Samme som udskrift, med 8 px Fusion-skrifttypen.

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

print10

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

Samme som udskrift, med 10 px Fusion-skrifttypen.

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

line

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

Linje fra x0,y0 til x1,y1. To argumenter fortsætter fra den forrige ende. line() nulstiller den kæde.

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

circ

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

Cirkelomrids. Radius er i pixels.

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

circfill

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

Udfyldt cirkel.

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

oval

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

Ellipseomrids inden for afgrænsningsboksen. Hjørner er inklusive.

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

ovalfill

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

Udfyldt ellipse i en afgrænsningsboks.

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

rect

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

Rektangelomrids. Hjørner er inklusive.

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

rectfill

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

Udfyldt rektangel.

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

rrect

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

Omrids af afrundet rektangel. x og y er øverst til venstre. w og h er størrelsen i pixels; begge skal være større end 0. r er hjørneradius og er som standard 0. Den er fastspændt til halvdelen af den korte side.

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

rrectfill

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

Udfyldt afrundet rektangel. Samme argumenter som rrect.

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

fillp

`fillp([p])`

16-bit udfyldningsmønster for senere udfyldte former. 0 er fast. fillp() returnerer det aktuelle mønster. Bits følger 4×4-mønsteret.

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

pal

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

Omdanner farve c0 til c1. Tilstand 0 er tegnepaletten, 1 visningspaletten, 2 udfyldningsmønsterpaletten. pal() nulstiller.

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

palrgb

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

Sætter én paletindgang til RGB, hver kanal 0..255.

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

palt

`palt([col, t])`

Gennemsigtighed. palt(col) eller palt(col, true) gør col gennemsigtig. palt(col, false) gør den solid. palt() gør alle farver uigennemsigtige.

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

setpalette

`setpalette(n)`

Vælger palette bank n. setpalette() returnerer den aktuelle bank.

```
setpalette(1)
```

tline

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

Tegner en linje på skærmen, mens der udtages kortpixels fra kortpositionen mx, my, mdx og mdy. Bevæger kortprøven trin for trin langs linjen.

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

Sprites og kort

spr

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

Tegner sprite n ved x, y. w og h er størrelsen i pixels, standard 8. fx og fy vender vandret og lodret, når de ikke er nul. tex er sprite-arkets faneblad. rot er vinklen i drejninger (0..1), samme retning som sin og cos. 0 efterlader spriten uroteret.

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

sspr

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

Kopierer et rektangel af sprite-arket til skærmen, skaleret til dw med dh. Standardindstillingerne bevarer kildestørrelsen.

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

sget

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

Farven på én texel på sprite-arket.

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

sset

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

Skriver én texel på sprite-arket.

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

fget

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

Sprite-flag. fget(n) returnerer flagbyten. fget(n, f) er sand, når flagbit f (0..7) er sat.

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

fset

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

Sætter sprite-flag. fset(n, v) skriver hele byten. fset(n, f, v) sætter én bit.

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

setspr

`setspr(n)`

Vælger det aktive sprite-ark. setspr() returnerer det aktuelle ark. Samme bank som settex.

```
setspr(1)
```

settex

`settex(n)`

Alias for setspr.

```
settex(0)
```

map

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

Tegner kortet. Celle cx, cy på kortet tegnes på skærm sx, sy. cw, ch på 0 betyder hele kortet. Laget accepteres og ignoreres. tex er flisesættet.

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

mget

`mget(cx, cy)`

Sprite-id i kortcelle cx, cy.

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

mset

`mset(cx, cy, n)`

Skriver sprite n ind i kortcellen cx, cy.

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

setmap

`setmap(n)`

Vælger kort n. setmap() returnerer det aktuelle kort. Kort 0 er et reelt kort.

```
setmap(1)
```

Hukommelse

peek

`peek(addr)`

Læser én byte af det 64 KB store vindue. Vinduet er live-skærmen, derefter sprite-pixels og mapper derefter celler, når skærmen passer ind i 65280 bytes. Større skærme pages med poke(0x5F40, page). Adresser, der ikke er mappet, returnerer 0. Skrives også @addr.

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

poke

`poke(addr, ...)`

Skriver bytes til addr. Ekstra argumenter skriver følgende bytes. Ikke-mappede adresser ignoreres. Skrives også $addr, val. At prikke skærmen tæller som en pixelskrivning.

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

peek2

`peek2(addr)`

Usigneret 16-bit værdi, little-endian, ved adresse.

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

poke2

`poke2(addr, val)`

Skriver en usigneret 16-bit værdi, little-endian.

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

peek4

`peek4(addr)`

Usigneret 32-bit værdi, little-endian.

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

poke4

`poke4(addr, val)`

Skriver en usigneret 32-bit værdi, little-endian.

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

memcpy

`memcpy(dst, src, len)`

Kopierer len bytes inden for det samme 64 KB vindue.

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

memset

`memset(dest, val, len)`

Fylder len bytes startende ved dest med val.

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

Matematik

abs

`abs(x)`

Absolut værdi.

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

flr

`flr(x)`

Runder mod minus uendelighed.

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

ceil

`ceil(x)`

Runder mod plus uendelighed.

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

max

`max(a, b)`

Størst af de to værdier.

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

min

`min(a, b)`

Den mindste af de to værdier.

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

mid

`mid(a, b, c)`

Den midterste af de tre værdier. Nyttig som klemme.

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

sgn

`sgn(x)`

Tegn for x: -1, 0 eller 1.

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

sqrt

`sqrt(x)`

Kvadratrod. Et negativt argument er ikke et tal.

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

sin

`sin(x)`

Sinus x går på skift: 0, 0,25, 0,5, 0,75 går én gang rundt om cirklen. Ikke radianer.

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

cos

`cos(x)`

Cosinus har til gengæld samme konvention som sinus.

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

atan2

`atan2(dx, dy)`

Vinkel for vektoren dx, dy, i sving fra 0 til 1.

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

rnd

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

Tilfældigt tal i [0, x). Standard x er 1, så rnd() er 0..1. For en terning, flr(rnd(6))+1.

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

srand

```
srand(seed)
```

Indsæt frø i den tilfældige generator, så en senere omgangssekvens kan gentages.

```
srand(1)
```

band

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

Bitvis OG. Også skrevet a & b.

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

bor

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

Bitvis ELLER. Også skrevet a | b.

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

bxor

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

Bitvis XOR. Også skrevet a ^^ b.

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

bnot

`bnot(a)`

Bitvis IKKE. Også skrevet ~a.

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

shl

`shl(a, n)`

Skift til venstre. Også skrevet som et << n.

shr

`shr(a, n)`

Aritmetisk forskydning til højre. Også skrevet som et >> n.

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

lshr

`lshr(a, n)`

Logisk skift til højre. Også skrevet som et >>> n.

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

rotl

`rotl(a, n)`

Roter bits til venstre. Også skrevet som et <<> n.

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

rotr

`rotr(a, n)`

Roter bits til højre. Også skrevet som >><< n.

Tabeller og værdier

add

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

Tilføjer v til tabel t. Hvis i er givet, indsættes ved det 1-baserede indeks.

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

del

`del(t, v)`

Fjerner den første post lig med v.

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

deli

`deli(t, [i])`

Fjerner indeks i og returnerer den værdi. Uden i fjernes den sidste indtastning.

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

all

`all(t)`

Iterator over arraydelen, i rækkefølge. Huller stopper gangen på den måde, et 1-baseret array slutter.

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

foreach

`foreach(t, fn)`

Kalder fn(v) for hvert array-element.

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

pairs

`pairs(t)`

Iterator over hver nøgle og værdi, inklusive navngivne felter.

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

ipairs

`ipairs(t)`

Iterator over array-delen med det 1-baserede indeks.

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

next

`next(t, [k])`

Returnerer den næste nøgle og værdi efter k. next(t) starter i begyndelsen. Bruges af par.

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

count

`count(t, [v])`

Længden af arraydelen. Hvis v er givet, tælles hvor mange poster der er lig med v.

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

pack

`pack(...)`

Pakker argumenterne ind i en ny tabel.

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

unpack

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

Returnerer t[i] til t[j] som separate værdier. Standardværdierne dækker hele array-delen.

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

select

`select(i, ...)`

select("#", ...) returnerer antallet af argumenter, der følger. select(i, ...) returnerer argumenterne fra i og fremefter.

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

assert

`assert(c, [msg])`

Stopper med en fejl, når c er falsk eller nul. msg erstatter standardteksten.

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

pcall

`pcall(f, ...)`

Kalder f. Ved succes returnerer `true` og resultaterne. Ved fejl returnerer `false` og fejlteksten.

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

Strenge

chr

`chr(n)`

Et tegn fra en bytekode 0..255.

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

ord

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

Bytekoder for s fra indeks i til j. Indekser er 1-baserede. Standardværdien er det første tegn.

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

sub

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

Delstreng fra i til j, 1-baseret. Udelad j for at tage halen.

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

split

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

Opdeler s i en tabel. sep bruger som standard komma. Et tal sep er en chunkbredde. conv konverterer brikker med tonum, når det er sandt.

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

tonum

`tonum(s, [base])`

Analyserer et tal. Grundtal kan være 2..36. Returnerer nul, når teksten ikke er et tal.

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

tostr

`tostr(v, [digits])`

Omdanner en værdi til tekst. cifre angiver decimaler for tal.

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

Lyd

sfx

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

Afspiller mønster n og returnerer kanalen. Udelader ch, eller send -1, og den første ledige kanal vælges. ch -2 stopper mønster n på hver kanal. n -1 stopper en kanal, eller hver kanal, når ch udelades. n -2 lader et loopmønster afslutte i stedet for at loope. sfx() uden argumenter returnerer -1.

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

sfxget

`sfxget(n, step)`

Returnerer en tabel med ét mønstertrin: [1] tonehøjde, [2] bølge, [3] lydstyrke, [4] effekt. Tonehøjden er 0..63 (C-0 til D#5). Bølgen er 0..15. Lydstyrken er 0..7, og 0 er lydløs. Effekten er 0..7.

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

sfxset

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

Skriver ét mønstertrin. Alle seks argumenter er påkrævet.

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

sfxhdr

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

Mønsterheader. sfxhdr(n) returnerer headertabellen. sfxhdr(n, key, val) angiver ét felt: speed, loop_in, loop_out, level, noise, crush, dtune, rverb, dampn, swing, steps. Tallene 0..10 vælger de samme felter i den rækkefølge. speed er TIC, ticks per step. Higher er langsommere. 15 er 120 beats per minut. 16-bit accepterer 1..255. Smallere synths snapper til deres TIC-liste.

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

sfxstat

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

Mønster-id afspilles på kanal ch, eller -1. Standardkanal er 0.

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

note

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

Spiller én tone uden et mønster. tonehøjde 0..63, bølge 0..15, volumen 0..7, effekt 0..7. ticks er hvor længe den holder, i de samme 120 Hz ticks som TIC. Udelad ticks, og den varer 15, ét standardtrin. Effekten kører over den tid, hvorefter kanalen bliver stille.

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

music

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

Starter sang n fra toppen. music(-1) stopper. Hvis man kalder music med det id, der allerede afspilles, lader det køre. music() returnerer det afspillende sang-id, eller -1. fade er millisekunder: sangen fader ind eller fader ud, når n er -1. maskebits, der er indstillet, er de kanaler, der bruges til sangen. 0 betyder alle otte. De andre kanaler forbliver frie til sfx.

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

musicstat

`musicstat()`

Tabel over den sang, der afspilles. sang og række er sang-id'et og rækken, eller -1 når der ikke afspilles noget. step og pos har otte tal. step[1] er kanal 0, step[8] er kanal 7. step er nodeindekset, eller -1 når kanalen er lydløs. pos er hvor langt noden er gået, fra 0 til 1. tick er hvor mange tick denne række allerede har spillet. played er hvor mange rækker der er startet siden music(), inklusive den der afspilles.

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

songget

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

Mønster-id gemt i en sangcelle. ch er 0..7. 255 betyder, at cellen er tom. Dette er live-kaldet; kodehintteksten viser stadig en ældre argumentrækkefølge.

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

songset

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

Skriver et mønster-id i en sangcelle. Brug 255 til at rydde cellen.

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

stat

`stat(n)`

Kun lyd. 16..23 er sfx-id'et på kanal n-16. 24..31 er musikmønsteret på kanal n-24. Ethvert andet n returnerer 0. Frame timing og RAM er perf og info, ikke stat.

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

Datatabeller

dget

`dget(t, row, col)`

Læser en celle i datatabellen t fra DATA-panelet. Række og kolonne starter ved 0.

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

dset

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

Skriver en celle. Brug en streng til en tekstkolonne.

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

dstr

`dstr(t, row, col)`

Cellen som tekst. Tal konverteres.

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

drows

`drows(t)`

Hvor mange rækker har tabel t?

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

dcols

`dcols(t)`

Hvor mange kolonner har tabel t?

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

Gem og præstationer

save

`save(k, v)`

Gemmer v under tasten k i den gemte fil. v kan være et tal, en streng eller en tabel over disse. Filen er komprimeret.

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

load

`load(k)`

Læser nøgle k. Returnerer nul, når nøglen mangler.

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

delsave

`delsave(k)`

Sletter nøglen k fra gemningen.

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

saveslot

`saveslot([name])`

Vælger den gemte fil i forfatterens mappe. Uden navn er spilleautomaten spillets titel. Returnerer 1, når filen allerede findes.

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

achieve

`achieve(i)`

Låser op for præstation i, fra 0 til 15. Gemmes lokalt. Et Steam-build sætter også ACH_i. En katalogsession rapporterer det til spillerens konto.

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

achieved

`achieved(i)`

1 når præstation i allerede er låst op.

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

achdef

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

Skål for præstation i, vist i 15 sekunder fra bunden af skærmen. spr er en valgfri 8×8-flise, tekst dens ark.

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

achclear

`achclear(i)`

Rydder præstation i i den lokale fil. En Steam ACH_i, der allerede var indstillet, forbliver indstillet.

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

Netværk

whoami

`whoami()`

Dit spiller-id er 0..15, mens en session er i gang. Offline er det -1.

```
id = whoami()
```

nhost

`nhost()`

1 hvis denne maskine er værten, ellers 0.

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

nplayers

`nplayers()`

Hvor mange live-id'er findes der i 0..15.

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

nlisten

`nlisten()`

Starter hosting af UDP og returnerer LAN-adressen som ip:port.

```
addr = nlisten()
```

njoin

`njoin(addr)`

Tilslutter ip eller ip:port.

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

nleave

`nleave()`

Forlader sessionen. Peers ser _mp_leave med why 0.

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

nsend

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

Sender en tabsbeskyttet besked. Navnet er hændelsen. Op til fire ekstra værdier følger. p fra 0 til 15 hvisken til den pågældende peer; udelad p for at sende til alle.

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

ncall

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

Samme form som nsend, men beskeden forsøges at sendes igen, indtil den bekræftes. Tre fejl i træk afbryder sessionen.

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

nlan

`nlan()`

Nuværende LAN ip:port efter nlisten.

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

nstun

`nstun()`

Din offentlige IP-port fra STUN. En gæst på et andet netværk kan forsøge at oprette forbindelse til det.

```
pub = nstun()
```

setturn

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

TURN-fallback når en direkte sti fejler. setturn() med tomme argumenter slår den fra. Adressen er host:port.

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

nlist

`nlist(game)`

Tabel over åbne sessioner for det pågældende spilnavn. Hver post indeholder adresse, spillere og navn.

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

nannounce

`nannounce(game)`

Hjerteslag, så denne vært vises i nlist for spillet. Kald den på en timer under værtsskabet.

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

nkb

`nkb([on])`

Adressefelt. nkb(1) åbner det, nkb(0) skjuler det, nkb() fortæller dig, om det er åbent. Op og ned ændrer et tegn, 0-9 eller et punktum eller et kolon. A skriver det. X sletter. Tastaturet skriver ikke i det.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

Tekst i adressefeltet, derefter ændres tegnene op og ned. nkbstr("") sletter teksten.

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

_mp

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

Du skriver denne funktion. Den kører, når en besked ankommer. `from` er afsender-id'et. `name` er begivenheden. Op til fire værdier følger.

```
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 denne funktion. En peer modtog id p. Den udløses også for dig; whoami() er det id.

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

_mp_leave

```
_mp_leave(p, why)
```

Du skriver denne funktion. Peer p til venstre. Hvorfor 0 er en normal forladelse. Hvorfor 1 er en mistet forbindelse.

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

_mp_update

`_mp_update()`

Du skriver denne funktion. Den kører hver 15. _update-ramme, omkring 4 Hz. Et godt sted at sende positioner.

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

_mp_fail

`_mp_fail()`

Du skriver denne funktion. Sessionen er død: værten forsvandt, eller ncall mislykkedes tre gange.

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

Coroutines

cocreate

`cocreate(f)`

Laver en coroutine fra funktion f. Den kører ikke før coresume.

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

coresume

`coresume(c, ...)`

Kører coroutine c indtil den giver eller returnerer. Ved succes: sand og de resulterende værdier. Ved fejl: falsk og fejlen.

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

costatus

`costatus(c)`

"suspenderet", "løbende" eller "død".

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

yield

`yield(...)`

Giver kontrollen tilbage til den, der kaldte kerneværdien. Værdierne bliver resultaterne af den pågældende kerneværdi.

```
yield(1)
```

Måle

info

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

Tegner den live HUD i _draw: fps (gennemsnit på 1 sekund), upd, draw, px, max, ram, peak, ramc og net, når en session er slut. info() bruger printmarkøren. info(col) angiver kun farven. info(x, y) eller info(x, y, col) placerer den. Tal bruger B, k eller m, så de passer til en lille skærm. Selve opkaldet tælles ikke i px.

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

showstats

`showstats([on])`

Samme HUD som info, tegnet automatisk ved 0, 0 efter _draw, ignorerer kamera og klip. showstats(0) skjuler det. showstats(1) eller showstats() viser det.

```
showstats(1)
```

perf

`perf(n)`

Et HUD-nummer, så du selv kan tegne det. 0 fps, 1 opdatering, 2 tegninger, 3 px, 4 maks. (tungeste billede i dette sekund), 5 ram, 6 ramc (ram plus den pakkede spilkassette), 7 maksimale ram siden spilkassetten startede.

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