PIX37-håndbok

Redigeringsverktøy, deretter kode. Offline-kopi av boken på pix37.party/docs.

Redaktør

PIX37 DEV er editoren. Du tegner, skriver og hører spillet her, og eksporterer deretter en spillkassett eller en frittstående versjon. Den kan også spille av en spillkassett, og den kan åpne PIX37.PARTY for å laste ned spillkassetter og bruke katalogen. PIX37 DEV EDUCATIONAL er den samme editoren, og den er gratis. Den har ingen versjoner, ingen eksport av spillkassetter og ingen tilgang til PIX37.PARTY. PIX37 Console er avspilleren. Den spiller av `.p37` -spillkassetter, og du åpner PIX37.PARTY for å laste ned spillkassetter og bla gjennom katalogen.

Prosjekter og spillkassetter

En `.pix37` -fil er prosjektet: kildefaner, sprite-ark, kart, lyd og datatabeller. En `.p37` fil er spillkassetten som spilleren kjører: tettpakket bytekode.

Lagringer og prestasjoner ligger i forfattermappen ved siden av appdataene. På Windows ser det slik ut: `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` for en lagring, og `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` for prestasjoner. På Linux er de samme filene `~/.local/share/PIX37/Mira/Pong.p37s` og `~/.local/share/PIX37/Mira/Pong.ach` . Ada er kontoen på datamaskinen. Mira er forfatternavnet, og det navnet er mappen. Pong er lagringssporet, som som standard er spilltittelen. `save` skriver `.p37s` . Den første opplåste prestasjonen skriver `.ach` . Hvis filene allerede er der, leser løperen dem når spillkassetten starter.

Utviklermeny

Escape åpner utviklermenyen. Escape mens et spill kjører, går tilbake til redigeringsprogrammet. Bekreft med Enter eller A-knappen (V som standard). Tilbake er B-knappen (C som standard).

Uten lastet inn noe prosjekt er menyen: OPPRETT PROSJEKT, LAST IN PROSJEKT, BYGG, LASTE IN spillkassett, KOBLE TIL PIX37.PARTY, INNSTILLINGER.

Med et prosjekt lastet er det: LAGRE PROSJEKT, LAST INNTAK I PROSJEKT, OPPRETT PROSJEKT, RESSURE, BYGG, LAST INNTAK I SPILLKASSETT, KOBLE TIL PIX37.PARTY, INNSTILLINGER.

Ctrl+S lagrer det åpne prosjektet. Ctrl+R kjører det.

Opprett et prosjekt

OPPRETT PROSJEKT ber om spillkassetten før du begynner å lage prosjektet.

- **VIDEO** — 1-BIT (2 farger), 2-BIT (4), 4-BIT (16), 8-BIT (256).
- **AUDIO** — sampledybde for synthen: 16, 8, 4, 2 eller 1 bit.
- **SKJERM** – forhåndsinnstillingene inkluderer 64×64, 128×128, 160×90, 256×256 og 320×180. Med TILPASSET størrelse kan du velge en hvilken som helst størrelse fra 1×1 til 512×512.
- **PALETTER** – fargebanker. Den første banken starter som settet med 16 farger nedenfor. LOAD PALETTE bruker et innebygd sett: PICO-8 16, PICO-8 32, Game Boy, 1-bit, CGA, Sweetie-16 og en lengre liste hentet 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 laste inn en palettfil, eller laste inn din egen PNG og ta paletten ut av det bildet.



Fargeindekser brukt av `cls` , `pset` , `spr` og resten. Du kan male en hvilken som helst indeks på nytt i palettredigereren.

De fem panelene

Redigeringsprogrammet er GAME, PAINT, CODE, SOUND og DATA, i den rekkefølgen. Kartet ligger inni PAINT. SFX og sanger ligger inni SOUND. Klikk på et panel for å jobbe der. Når CODE har fokus, ignorerer SOUND og DATA bokstavtaster, slik at skriving ikke også spiller piano.



Full layout, skrevet `1:1:4:1:1` . Hvert tall angir hvor mye av raden panelet tar: SPILL, MALING, KODE, LYD, DATA. En liten skjerm sitter sentrert i en celle på 128 × 128. En skjerm som er større enn 128 piksler utvider SPILL-kolonnen.

INNSTILLINGER → GLOBAL → LAYOUT sykler gjennom arrangementene:

Kode	Det du ser
`1:1:4:1:1`	Alt. SPILL og LYD stablet, MALING og DATA stablet, KODE til høyre.
`1:1:4:1:1 2X`	Samme oppsett. KODE tegnes med dobbel tetthet, slik at dobbelt så mye tekst får plass.
`1:0:1:0:1`	SPILL, DATA, KODE.
`1:0:1:0:1 2X`	Samme arrangement, med den tettere KODEN.
`0:1:1:1:1`	Ingen forhåndsvisning av spillet. MALING, KODE, LYD, DATA.
`0:1:1:1:1 2X`	Samme arrangement, med den tettere KODEN.
`0:1:4:1:2`	Studio. Bred CODE, en høyere DATA-stripe, MALING og LYD.
`0:1:4:1:2 2X`	Samme arrangement, med den tettere KODEN.
`0:0:4:0:4`	Kun KODE og DATA, begge store.
`0:0:4:0:4 2X`	Samme arrangement, med den tettere KODEN.
`0:1:4:1:0`	MALING, LYD og en bred KODE.

`0:1:4:1:0 2X`	Samme arrangement, med den tettere KODEN.
`0:4:0:0:0`	Kun MALING, 256×256, én piksel av arket til én piksel av lerretet.

En `2X` rad beholder de samme panelene. CODE-panelet tegnes med dobbel oppløsning og skaleres tilbake til samme celle, slik at teksten, fanene og innermargen er halvparten så høye og dobbelt så mye kilde får plass på skjermen.

SPILL

Forhåndsvisningen av den kjørende spillkassetten. Klikk på SPILL-panelet, eller trykk Ctrl+R, for å kjøre forhåndsvisningen. INNSTILLINGER → SPILLPANEL er der panelet konfigureres:

- **Forhåndsvisning** – inaktiv farge, kontinuerlig eller automatisk innlasting når du redigerer.
- **Spill i et nytt vindu** – spillet åpnes ved siden av redigeringsprogrammet.
- **Skjerm** – forhåndsvisning som RGB888 eller RGB565.
- **SPILL 64** — en 64×64-skjerm er enten sentrert 1:1 i cellen, eller skalert 2× for å fylle den.
- **Shaders** — CRT og filtre i spillet, separat fra filtrene i redigeringsprogrammet.
- **Farge** — bakgrunnen til panelet.

MALING

Sprite-ark og, på kartsiden av det samme panelet, flisede kart. Ark kan være større enn skjermen. Størrelsene på sprite-forhåndsvisningen er 8, 16, 32, 64 og 128 (128 bare når arket er minst 128 piksler). Bla med musehjulet over sprite-forhåndsvisningen for å zoome den inn. Penselstørrelsene er 1, 2, 3 og 4.

Nøkkel	Verktøy
B	Blyant
M	Rektangelvalg
Q	Lasso
S	Stemple
G	Bøtte
jeg	Pipette
V	Flytte
L / Shift+L	Linje / kurve
U / Shift+U	Hult rektangel / hul ellipse

F / Shift+F	Fylt rektangel / fylt ellipse
D	Dither
[ ]	Mindre/større børste
Ctrl+Z / Ctrl+Y	Angre / gjør om. Ctrl+Shift+Z gjør også om.
Ctrl+C / Ctrl+X / Ctrl+V	Kopier, klipp ut, lim inn
Ctrl+A	Velg lerretet

Kartet bruker samme idé, med færre verktøy: B blyant, M rektangelvalg, S stempel, V flytting og Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Hold inne Alt for å zoome inn i karttekstureN. Penselstørrelsen på kartet forblir med maleverktøyene.

Hver sprite har åtte flaggbiter. Paint setter dem på spriten; `fget` og `fset` leser dem i spillet.

INNSTILLINGER → MALINGSREDIGERING er der panelet konfigureres:

- **Palett** – hvilken bank som tegnes, pluss lasting og lagring.
- **Forhåndsvisning** — RGB888 eller RGB565.
- **Penselmarkør** – pekeren mens du maler.
- **Pikselsnapp** – hvordan penselen snapper til piksler.
- **Stor børste** – en stor børste knipser fra øverste venstre hjørne, eller fra hele fotavtrykket.
- **Løkskall** — et spøkelse fra andre rammer, og hvilken retning spøkelset vender mot.
- **Kartzoom** – ekte piksler, eller en fri zoom.
- **Farger** – fargene i malingsgrensesnittet.

KODE

Faner er separate filer inne i prosjektet. Klikk på et fanenavn for å gi det nytt navn; Enter bekrefter. Flere faner som hver definerer `_init` , `_update` eller `_draw` er sydd sammen i tabulatorrekkefølge: den første fanen kjøres først, den siste fanen kjøres sist. Kode som ikke er inne i en hook blir en del av `_init` . En spillkassett uten hooks i det hele tatt behandles som én `_draw` .

Nøkkel	Handling
Piler	Flytt cirkumfleksknappen. Shift utvider det merkede området.
Ctrl+Venstre / Høyre	Med ord.
Hjem / Slutt, Ctrl+Hjem / Slutt	Starten eller slutten av linjen, eller av tabulatoren.

Fane	Sett inn to mellomrom. Godta navnet i fullføringslisten.
Gå	Ny linje, innrykket som den forrige. Godtar en fullføring.
Ctrl+C / X / V	Kopier, klipp ut, lim inn.
Ctrl+Z / Ctrl+Y	Angre / gjør på nytt.
Ctrl+A	Velg fanen.
Ctrl+D	Dupliser linjen eller det markerte området nedover.
Ctrl+/	Kommenter eller fjern kommentar med `--` .
Ctrl+F	Finn. F3 hopper til neste treff. Escape lukker linjen.
Ctrl+G	Gå til et linjenummer.
Flykte	Lukker søk, gå til eller hintlisten først. En andre Escape-tast forlater redigeringsprogrammet for utviklermenyen.

Hint dukker opp mens du skriver inn et funksjonsnavn. Enter eller Tab setter inn navnet, og et `(` hvis FULLFØRT (er på i innstillingene. INNSTILLINGER → KODEERDIKTERING:

- **Søk** — Ctrl+F i den aktive fanen, eller på tvers av alle faner.
- **Tekst høy** – beskjær tegn til 5 piksler, eller vis hele tegnet.
- **Strengstørrelse** — skriftstørrelsen til strengliteraler.
- **Kommentarfont** – 3×5, eller Fusion-fonten til gjeldende brukergrensesnittspråk.
- **Fullført (** — om fullføring skriver inn parentesen.
- **Kodetips** – av eller på.
- **Hintspråk** — menyspråket, eller engelsk.
- **Cirkumfleksavstand** – ekstra forskyvning på tomme linjer.
- **Innmarsj** — linjetall.
- **Farge** — bakgrunn, kommentarer, tabulatorer, tall, operatorer, nøkkelord, strenger, søkemotorfunksjoner, dine funksjoner, variabler, parenteser, cirkumfleks, valg, feillinje, advarsel.

LYD

Panelet har to faner. SFX er ett mønster: en kort rekke med noter. SOUND er en sang: den plasserer disse mønstrene på åtte kanaler. TIC er hvor mange tikk ett trinn venter. Klokken er 120 tikk i sekundet, og ett trinn er en sekstendedelsnote, så 15 tikk er 120 slag per minutt. En høyere TIC er et langsommere mønster. To mønstre holder seg i takt når den ene TIC-en deler den andre: 15 og 30 gjør det, 15 og 8 gjør det ikke.

Pianoet spiller når en note velges med musen. Hvite tangenter er ZXCVBNM (CDEFGAB). Svarte tangenter er SDGHJ (C# D# F# G# A#). Hold musepekeren over en hvilken som helst verdi og skriv inn et tall, eller hold

musepekeren over den og bla med hjulet. Tonetall i kode går fra 0 (C-0) til 63 (D#5).

På ett mønster er LOOP og LEN de samme to tallene. OUT over 0 går i løkker fra IN til OUT, og tittelen er LOOP. OUT ved 0 og IN over 0 er LEN: så mange noter spilles fra starten, så slutter mønsteret, og tittelen er LEN. Begge ved 0 spiller hvert trinn.

Hvor mye av synthen du kan redigere avhenger av bitdybden som er valgt for spillkassetten. INNSTILLINGER → SFX EDITOR angir dybden, om en note automatisk går videre (av, notekjede eller feltkjede), om Space starter avspillingen på nytt eller veksler mellom start og stopp, og redigeringsprogrammets farger.

16-bit

Hvert trinn har en tone, et volum, en bølge og en effekt. Volumet er fra 0 til 7. 0 er stille.

Bølger:

- **0 trekant**
- **1 skråstilt sag**
- **2 sag**
- **3 kvadrat**
- **4 pulser** — en smal puls
- **5 orgel**
- **6 støy**
- **7-fase**
- **8 PWM** — en firkant hvis bredde beveger seg
- **9 pulser 25 %**
- **10 sinus**
- **11 metallisk støy**
- **12 FM-klokke**
- **13 napp**
- **14 supersag**
- **15 PWM (C64)**

Effekter på en enkelt tone:

- **0 ingen**
- **1 lysbilde** – tonehøyde og volum glir til neste note over dette trinnet
- **2 vibrato** — tonehøyde-svingninger
- **3 dropp** — tonehøyden faller til stillhet, det vanlige sparket
- **4 fade inn**
- **5 fade ut**
- **6 arpeggioer** — går denne noten og de neste tre, én gang for hver 8. tikk. Ved TIC 8 eller lavere, hver 4. tikk.
- **7 raske arpeggioer** — de samme fire tonene, hvert 4. tikk, eller hvert 2. når TIC er 8 eller lavere.

Mønsteroverskriften, over notene:

- **TIC** — tikker per trinn, fra 1 til 255. 15 er 120 slag per minutt. Skriv inn tallet, eller steg det ett trinn i taget.
- **LOOP INN / UT** – med OUT over 0, går mønsteret i en løkke fra IN til OUT. Hvis OUT står på 0, blir tittelen LEN: IN er hvor mange noter som spilles, og 0 betyr hele mønsteret.
- **STØY** — hele mønsteret blir støy.
- **CRUSH** — en livlig variant av hver bølge.
- **DTUNE** — 0 av, 1 et lett refreng, 2 et sterkt refreng.
- **RVERB** — etterklang: 0 av, 1 kort, 2 lang.
- **DAMPN** — et lavpassfilter: 0 av, 1 lys, 2 sterk.
- **LEVEL** — volum for hele mønsteret, 0 til 7.
- **SVING** — stokking. Rare trinn venter litt lenger, til og med trinn litt kortere.
- **TRINN** — hvor mange trinn mønsteret bruker: 32, 24, 16 eller 12.

8-bit

Et trinn har en note og en bølge. De fire bølgene er firkant, trekant, sag og støy. Toppteksten har TIC, LOOP IN/OUT, FX og LEVEL. TIC er én av 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 trinn er bare en note. Overskriften har TIC, LOOP (på eller av), WAVE (de samme fire bølgene) og OCT (hvilken oktav notene sitter i). TIC er en av 4, 6, 8, 10, 12, 15, 20, 30.

2-bit

Et trinn er tomt, en tromme, et treff eller en knusing. Toppen har TIC og BASE. TIC er 8, 15, 16 eller 30. BASE angir hvor høyt trommen og treffet sitter.

1-bit

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

Sanger

LYD-fanen er den åpne sangen. Åtte kolonner er kanalene. En celle er en mønster-ID. 255, tegnet tom, betyr at kanalen er stille på den raden. X-kolonnen er en kommando: fortsett, stopp, løkkestart eller løkkeslutt. Stopp avslutter sangen etter raden. Løkkeslutt hopper tilbake til siste løkkestart.

En rad slutter når mønsteret lengst til venstre med LØKKEN UT på 0 er ferdig. Mønstre lenger til høyre gjør ikke den raden lengre: et lengre mønster klippes, et kortere starter på nytt på sitt første trinn til raden slutter.

Et mønster med LØKKEN UT over 0 går i løkker mellom LØKKEN INN og LØKKEN UT og avslutter ikke raden. Hvis alle mønsterene på raden går i løkker, forblir raden der.

Snarveier

Nøkkel	Handling
Pianotaster	Skriv et notat om trinnet som er valgt med musen.
Opp / Ned	Endre verdien under markøren.
Rom	Spill av den åpne SFX-en, eller den åpne sangen fra første rad. Start/stopp eller start på nytt, fra SFX EDITOR-innstillingene.
Ctrl+Z / Ctrl+Y	Angre / gjør på nytt.
Ctrl+C / Ctrl+X / Ctrl+V	Kopier, klipp ut, lim inn utvalget.

DATA

Navngitte tabeller. Hver tabell består av rader og kolonner. Fra koden bruker `dget` , `dset` , `dstr` , `drows` og `dcols` disse tabellene. Rad- og kolonnenumre i koden starter på 0.

Kolonnetyper, syklet i overskriften:

- **bool** — én bit.
- **2** — to biter.
- **4** — fire biter.
- **8** — en byte.
- **16** — 16 bits, for ID-er og større tall.
- **spr** — en sprite-ID.
- **str** — tekst, tegnet med systemfonten.
- **str8** og **str10** — samme tekst, tegnet høyere.
- **f16** og **f32** — en halvpresisjonsflåte og en fullflåte.

Når pekeren er på en kolonneoverskrift, endrer Ctrl+skroll kolonnens type. En vanlig rulling beveger seg over kolonner eller nedover rader, avhengig av hva som velges av INNSTILLINGER → DATAREDIGERING. Ctrl bytter den retningen, og det er derfor Ctrl+skroll på en overskrift endrer typen når hjulet er satt til kolonner.

Piltastene beveger seg. Enter redigerer en celle. INNSTILLINGER → DATAREDIGERING:

- **Hjul** – bla gjennom kolonner eller rulle gjennom rader.
- **Farger** – rutenettet, overskriften og utvalget.

Eiendeler

ASSETS importerer og eksporterer den delen som er åpen, ikke hele prosjektet. Sprites bruker spriten som er åpen i PAINT, på arket som er åpent. Kartet er kartet som er åpent. SFX er mønsteret som er åpent. Musikk er sangen som er åpen. Data er tabellen som er åpen. Kode er unntaket: hver tabulator skrives, og hver tabulator leses tilbake.

Type	Fil	Hva som beveger seg
Sprites	`.png`	Den åpne spriten.
Kart	`.p37map`	Det åpne kartet.
SFX	`.p37sfx`	Det åpne mønsteret.
Musikk	`.p37mus`	Den åpne sangen. Mønsternumrene forblir som de er. Eksporter SFX separat hvis du trenger disse lydene.
Data	`.p37dat`	Det åpne bordet.
Kode	`.lua`	Hver fane.

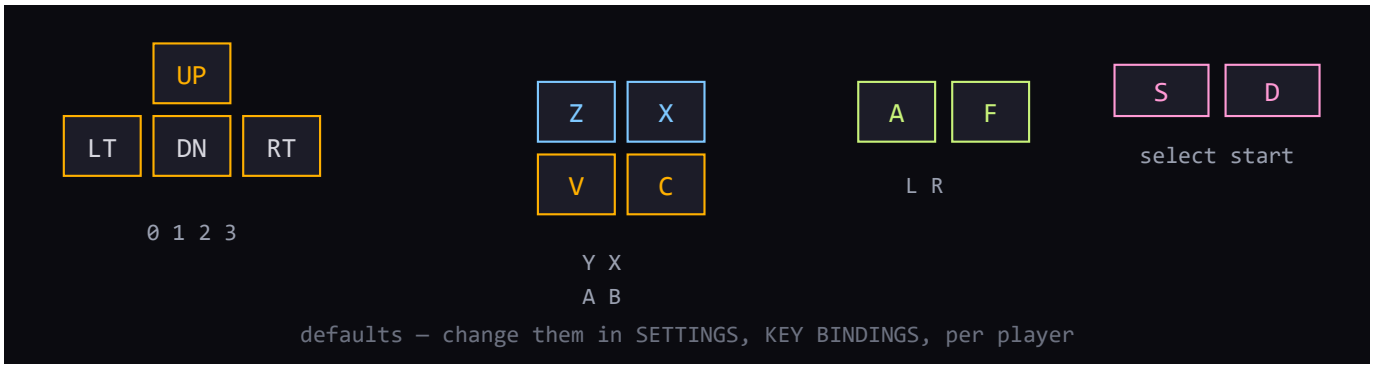
Innstillinger

Rotlisten:

Side	Hva den lagrer
PROFILER	Lagre og last inn alle innstillinger. Ny, gi nytt navn, overskriv, slett. Kopier eller lim inn profilen som tekst.
Meny	Vindu, taster, menyfarger, valgfri bakgrunn.
SPILLPANEL	Forhåndsvisning og avspillingsvinduet.
KODEREDITOR	Utheving og hint.
MALINGSREDIGERING	Pensel, palett, kartvisning.
SFX-REDAKTØR	Synthesizer og noteinntasting.
DATAREDAKTØR	Hjulretning og farger.
SPRÅK	Meny- og hintspråk.
LISENSER	Lisenstekster, inkludert Fusion Pixel-skrifttypen og SDL3.

GLOBAL:

- **OPPDELING** — paneloppsett, ovenfor.
- **VOLUM** — masternivå, 0 til 10.
- **TASTBINDINGER** — taster for hver av de 8 spillerne. Trykk på en tast for å fange den. Instruksjoner, ABXYLR, velg.
- **Presenter** – hopp over inaktive rammer, eller tegn alle rammer.
- **Fullskjerm** – uten kantlinjer.
- **Oppdater** — vindu FPS. Venstre og høyre trinn 60, 90, 120 og maks. Skriv inn et tall opptil 1000. 0 betyr ingen grense.
- **Vsync** — vent på visningen mellom rammene.
- **Skala** — heltallsvindumultiplikator.
- **Shaders** — CRT og filtre i editoren, ikke i spillet.
- **Farge** — menytekst og utheving.
- **Bakgrunn** – en valgfri PNG, beskåret til 128, bak menyen. Fjerner den.
- **Redigeringsmarkør** — pekerform.
- **Mus** — flytende peker, eller festet til piksler.
- **UI-hastighet** – hvor raskt en holdt retning gjentas gjennom en liste.
- **Utforsker** – fillister starter ved datamaskinens rot, eller i mappen til exe-filen.



Standardtastatur. A er bekreftelse i menyer. B er tilbake. I et spill er dette knappene 4 og 5. Velg (S) er knapp 10, ikke en menysnarvei.

Knapp	Rolle	Standardnøkkel
0	Igjen	Venstre pil
1	Høyre	Høyre pil
2	Opp	Pil opp
3	Ned	Pil ned
4	A, bekrefte	V
5	B, tilbake	C

6	X	X
7	Y	Z
8	R	F
9	L	EN
10	Velge	S
11	Start	D

Bygge

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

- **Navn** – filnavnet på bygget.
- **Forfatter** — på Windows og Linux. Dette er mappenavnet for lagrede filer.
- **Steam-app-ID** – på Windows og Linux. Tom betyr ingen Steamworks. En utfylt ID tilordner `achieve(0)` … `achieve(15)` til Steam-prestasjoner ACH_0…ACH_15.
- **Id** – bunt-identifikator der plattformen ber om en.
- **Skjul kode** – på overlegg på datamaskin og mobil. Det reduserer `peek` av spillkassetten. Det er ikke en lås.
- **Fullskjerm** – den eksporterte spilleren starter i fullskjerm.
- **Shaders** – opptil tre lag innebygd i spilleren.
- **Last ned .p37** – Kun for nett. Viser en nedlastingsoppføring i pausemenyen.
- **Ikon** — en flis fra et sprite-ark: ark, størrelse, x, y. Eller et PNG-omslag du beskjærer.
- **Eksporter** – velg mappen. Menyen viser BYGG LAGRET eller BYGG MISLYKKET.

Lag en build med internett på. Splash-en kan spørre serveren anonymt om hashen til denne builden er kjent. Den sjekken er der av to grunner. Den beskytter meg, forfatteren av PIX37, mot kopier av verktøyet som ikke var mine å gi bort. Den beskytter også personen som lagde spillet. Å ta andres spill og liste det opp som ditt eget blir stadig vanligere på telefonmarkedet, for eksempel Play (Android) eller iOS. Spillet starter fortsatt. Hvis maskinen er frakoblet, sier introen at spillet ikke er tilkoblet. Den linjen er også et hint til spilleren om at kopien kanskje ikke kommer fra en lovlig kilde. Det er en liten sjekk, slik at hvis spillet ditt ble stjålet og solgt uten ditt samtykke av noen andre, kan du gjøre det i en butikk som Play (Android), Steam eller itch.io. Serveren lagrer tidspunktet da hashen først ble registrert, ved siden av hashen, og endrer den ikke senere. Det tidligere tidspunktet er den opprinnelige builden. En hash registrert etter det er en senere kopi.

Jeg vil at denne kontrollen skal beskytte de som lager spill. Spillmotoren gjør allerede mye av jobben for dem, og dette er en av de tingene. Jeg forstår at det kan være kontroversielt. Hvis fellesskapet bestemmer seg for at det er bedre å droppe denne mikro-DRM-en, vil jeg gjøre det.

En gammel spillkassett fra en tidligere editor kan nekte å starte opp etter at språket er endret. Åpne `.pix37` -prosjektet i denne editoren og eksporter en ny `.p37` .

Koble til PIX37.PARTY

KOBLE TIL PIX37.PARTY er inne i appen. Det finnes ingen innloggingsside i nettleseren. Du logger deg på med en Steam-øktbillett eller med itch.io, ved hjelp av tastaturet og musen. Tjenesten viser et unikt visningsnavn, 2 til 16 tegn. Lisensnøkkelen og Steam-ID-en vises ikke til andre spillere.

Etter tjenestevilkårene er knutepunktet Favoritter, Katalog, Rangering, Spillere og profilen din.

En utviklerlisens er redaktøren. Du lager spillet, du laster det opp, og du legger igjen stjerner. En spillerlisens er spilleren. Hjerter er fra folk som spiller på en runner.

- **Katalog** – bla gjennom spill. Filtre dekker sjanger (opptil tre), skjermstørrelse, spillkassettstørrelse og om opplastingen inkluderer et `.pix37` -prosjekt.
- **Favoritter** – spill du har merket.
- **Opplasting** – krever en utviklerlisens. Du sender en `.p37` , og du kan legge ved `.pix37` -prosjektet. En `.p37` er pakket bytekode. Den er vanskelig å lese, og den kan bli tilslørt med vilje. En `.pix37` er prosjektfilen du faktisk jobber i: kilden, bildene og lyden, i en form en annen person kan åpne og lære av. Ved å sende den lar du andre skapere studere spillet i stedet for en komprimert spillkassett. Tittel, forfatter, sjangere, omslag, ikon og prestasjonsnavn følger med. RAM-nummeret som sendes opp er toppen spillkassetten nådde.
- **Kommentarer og prestasjoner** – på spillsiden. Opplåsing av en prestasjon mens du er logget inn lagres på den kontoen så vel som på maskinen.
- **Rangering** – utviklere, sortert etter stjerner eller hjerter. Hver rad viser begge tellingene. Åpne et navn for å se den utviklerens spill, sortert etter hjerter.
- **Spillere** – kontoer sortert etter opplåste prestasjoner.

Kode

Språket er et lite Lua-stilskript. Denne halvdelen starter fra et tomt prosjekt, lærer språket og viser deretter et kort eksempel på hver funksjon.

Et første program

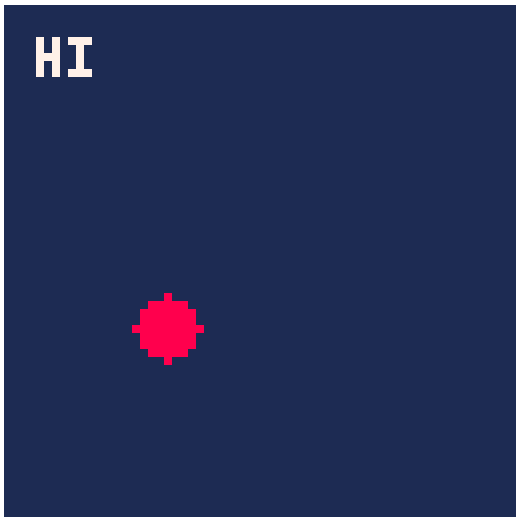
Opprett et prosjekt med 64×64, 4-bits farge. Åpne CODE. Erstatt tabulatoren med:

```
_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 kjører den. Piltastene flytter sirkelen. Escape stopper spillet og går tilbake til redigeringsprogrammet.



De samme tre kallene: `cls(1)` maler den marineblå skjermen, `print` tegner HI, og `circfill` tegner den rosa sirkelen.

Hva som kjører, og når

- `_init` kjører én gang.
- `_update` kjører på 60 Hz. Plasser bevegelse og knapper her.
- `_draw` maler. Uten fps-argument maler den så raskt som datamaskinen og skjermen kan. `_draw(30)` begrenser malingen til 30 fps og lar logikken være på 60.
- `_set_fps(n)` endrer den grensen senere. 0 fjerner den.

Navn skiller ikke mellom store og små bokstaver; editoren lagrer dem med små bokstaver. `function` og `fun` er det samme nøkkelordet.

Språk

Grunnleggende programmering

Et program er en liste over linjer som spillkassetten kjører. Hver overskrift viser først formen på linjen, deretter et lite eksempel. En linje som starter med `--` er en kommentar: en merknad til deg. Spillet hopper over den.

Variabler

En variabel er et navn som inneholder en verdi. Du velger navnet. Senere linjer kan lese det, og de kan lagre en ny verdi med samme navn.

Formen er `name = value` . `local name = value` holder navnet inne i gjeldende blokk, slik at resten av spillkassetten ikke kan se det. Uten `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` betyr at det ikke finnes noen verdi. `true` og `false` er yes og no. Et navn du aldri har angitt leses 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 inneholder mange verdier under ett navn. En liste bruker tall. Det første elementet er 1, ikke 0. En post bruker navn. En tabell kan inneholde en annen tabell.

Listeform: `{ value, value }` . Postform: `{ name = value, name = value }` . En liste over poster er en tabell der elementene 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 kropp

En betingelse er et spørsmål med et ja- eller nei-svar, for eksempel `hp > 0` eller `btnd(4)` . En kropp er gruppen av linjer som går når du går inn i en blokk. `if` og every løkker bruker samme form: en betingelse, deretter en kropp, deretter `end` . `repeat` avsluttes med `until` i stedet for `end` .

Hvis

`if` kjører bare kroppen sin når betingelsen er sann. `elseif` prøver en annen betingelse. `else` kjører når alle betingelsene er usanne.

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
```

Looper

En løkke gjentar en kropp. `while` sjekker betingelsen først og fortsetter så lenge den forblir sann. `repeat` kjører kroppen, deretter sjekker, slik at kroppen alltid kjører minst én gang. `for` tellinger. Det siste tallet er inkludert. Uten et tredje tall er trinnet 1, så hvis starten allerede er forbi slutten, kjører ikke kroppen. Nedtelling trenger et negativt trinn.

```
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 langs et bord

`all` besøker hvert element i en liste, fra det første. `pairs` besøker hvert navn og verdien som er lagret under det navnet.

```
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
```

Funksjoner

En funksjon er en navngitt oppskrift. Formen er `function name(arguments)` , deretter en kropp, og så `end` . Du kaller den ved å skrive navnet og parenteser. `return` sender en verdi tilbake, og den kan sende mer enn én. Argumentene er lokale verdier for den funksjonen.

```
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 funksjon som kan se de lokale verdiene rundt seg. Det er en avslutning. En tabell kan inneholde én, og du kaller den med en prikk.

```
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, nei, og å velge

snur `not` en betingelse. `and` er bare sann når begge sider er sanne. `or` er sann når begge sider er sanne. `a and b or c` velger `b` når `a` er sann, ellers `c` . `cond ? a : b` er den samme ideen på én linje: `a` når betingelsen er sann, `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)
```

Matematikk og tekst

`+ - * /` er addering, subtraksjon, multiplikasjon og dividering. `//` dividerer og beholder hele tallet. `%` er resten. `^` er potens. `..` limer sammen to tekster. `#t` er hvor mange elementer som er i en liste. Sammenligninger er `==` (samme), `~=` eller `!=` (forskjellig), og `< > <= >=` . `x += 1` legger 1 til x. Den samme formen 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
```

Der en lokal bor

`local` varer til `end` som lukker blokken sin. `if` / `end` er sin egen blokk. Du kan nevne en local over linjen som setter den, men inntil den linjen kjører er verdien `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 dit, men bare innenfor samme funksjon.

```
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` og `poke` -titt på levende piksler gjennom én 16-bits adresse. De beholder ikke en ekstra kopi av spillkassetten. Adressene `0x5F00` til `0x5FFF` er registre: kamera, klipp (skrivebeskyttet), tegnefarge, fyllmønster, de tre paletttilordningene og sidetallene. Bytene på begge sider av det hullet, 65280 av dem, er pikselvinduet.

Når skjermen får plass til disse 65280 bytene, 64×64 og 128×128 gjør det, og 256×256 ikke gjør det, starter skjermen på adresse 0, rad for rad. Det som er igjen er sprite-arket valgt av `0x5F41` (0 er ark 0). Hvis det arket er større enn den gjenværende plassen, velger `0x5F42` siden, og kartet er ikke i vinduet. Hvis arket får plass, er halen kartceller fra kartet valgt av `0x5F43` , én eller to byte hver, og `0x5F44` viser kartet når det ikke får plass i halen.

Når skjermen ikke får plass, er hele vinduet én side på 65280 piksler. `poke(0x5F40, n)` blar om på skjermen. `poke(0x5F41, n)` med n annet enn 0 viser ark n i stedet, og ark 0 har ingen visning i denne modusen. `poke(0x5F43, n)` med n annet enn 0 viser kart n, og det vinner over arket. Kart 0 i denne modusen er bare `mget` og `mset` .

Et `peek` som ikke lander på en piksel eller et register returnerer 0. Et `poke` der gjør ingenting. Sprite-flagg, SFX og DATA-tabeller er ikke i dette vinduet. Bruk `fget` , `sfxget` og `dget` .

Praktisk bruk

Hver funksjon er oppført nederst på denne siden, med praktisk bruk av PIX37-funksjoner. Lenkene under hvert eksempel åpner mer om funksjonene.



Jevn bevegelse

En spillkassett på 128×128, sett ovenfra. Sprite 1 er allerede tegnet. Pilene akselererer i alle retninger, og friksjon bremser deg. Spriteen brytes: en halvdel som forlater bunnen tegnes øverst, og det samme gjelder de andre kantene.

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



Sjekker om neste trekk er blokkert

En spillkassett på 64×64. Gulvet og den høyre veggen er allerede på kartet, malt med sprite 3, og flagg 0 er satt på den sprite. Piler beveger sprite 1. Ordet øverst sier blokkert når neste trinn ville overlappe en heltrukket flis, og fritt når det ikke ville det.

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



En klut med flip

Trykk A. Visningen tegnes med flip(), én svart rad om gangen, i stedet for inni _draw(). _draw holder seg unna mens det kjører, så hver rad forblir på skjermen.

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



En tabell med funksjoner (klasse)

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

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



Fire måter å lese en knapp på

A er knapp 4. Hold den inne og ladingen klatrer. Slipp den og ladingen går tilbake til 0. Opp og ned gjentas mens du holder den inne, slik at menyraden fortsetter å bevege seg. btnd er den første rammen av et trykk, som denne spillkassetten kun bruker som starten på det holdet.

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



Skyt med musen

Sprite 1 er allerede tegnet i midten. En hvit prikk er markøren. Venstreklikk skyter en piksel mot den. Hjulet endrer hastigheten, trykt i hjørnet.

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

Skriv inn et navn og bekreft det

Trykk A for å åpne feltet. Opp og ned endrer en blinkende bokstav. A skriver den, X sletter, Y er mellomrom, og B bekrefter. Linjen sier deretter navnet ditt.

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

Kamera på spilleren

En spillkassett på 128×128. Sprite 1 starter midt på et gulv med fliser. Pilene beveger seg. Kameraet holder spriten sentrert. x og y i hjørnet ruller ikke, fordi camera() uten argumenter setter visningen tilbake for HUD-en.

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

Noen få linjer fra en visuell roman

Linjene er i DATA, på engelsk og polsk. A fortsetter, eller velger blir værende når et valg vises på skjermen. B velger forlater språket ved et valg, og bytter ellers språk. X bytter språk mens et valg vises. Dialogen bruker 8px-skriften, slik at polske bokstaver tegnes.

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

Mal med titt og stikk

En spillkassett på 64×64, så adresse 0 er pikselet øverst til venstre og neste rad starter på 64. Spillkassett skriver piksler, kopierer en rad, tømmer en rad og skriver ut verdiene den leser tilbake.

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

En liste du kan legge til

A legger sammen et tall, opptil 8. B fjerner det siste. X fjerner det første. Listen skrives ut på skjermen.

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

Et navn tatt fra hverandre

Spillkassetten deler ada,12,blå og skriver ut navnet, den første bokstaven, 12, og ordet blå.

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

Spill en note, og endre den deretter

Mønster 0 er én note langt, og hver avspilling har samme lydstyrke. A spiller det igjen når den noten er ferdig. B hever tonehøyden og spiller den noten.

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

 En sang skrevet inn i spillkassetten

To mønstre går i løkke på rad 0, ett på hver av de to første kanalene. Raden forblir 0. ch0 og ch1 er tonen hver kanal er på. Å redigere mønsteret, LEN og LOOP i spillkassetten er en måte å se hva disse kontrollene gjør i musikkpanelet.

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

 Tre monstre i DATA

DATA-panelet har allerede en navnekolonne og en hp-kolonne, med slim, flaggermus og øye. A skader slimet. Tallene til høyre er dens hp.

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

 Husk poengsummen

A legger til et poeng og lagrer det. Avslutt og åpne spillkassetten igjen: nummeret er fortsatt der. B glemmer det.

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

  Et andre kapittel i en annen fil

To spillkassetter deler lagringsspor-oppdraget. Del 1: A markerer spillet som fullført. Del 2, som åpnes først, ber om å fullføre del 1. Etter det åpnes del 2, og B sletter lagringen slik at del 1 kan spilles på nytt.

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

 Én prestasjon

Sprite 8 er allerede trukket. Trykk A ti ganger. Skålen navngir prestasjonen, og sprite 8 forblir fordi den er opplåst. B fjerner prestasjonen i den lokale filen og nullstiller telleren.

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

 Et rom du kan gå inn i

En spillkassett på 256×256. En hvit prikk er markøren. Menyen er vert, bli med eller serverlisten. Verten åpner et rom med navnet gå og starter spillet. Bli med ruller gjennom en adresse (sifre, en prikk, et kolon) og B

kobler seg til. Serverlisten beveger seg opp og ned, og A blir med i raden. I spillet går pilene mot sprite 1. Én mynt deles: verten flytter den, og når man plukker den opp, legger det 1 til spillerens poengsum og den neste gyter. Hver sprite, inkludert din, viser poengsummen ved siden av. Hjørnet viser hvor mange spillere som er i, LAN-adressen og vertens offentlige adresse. B forlater: du og de andre går tilbake til menyen, og den neste verten starter et nytt rom med poengsum 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 venter

A viser neste linje. Funksjonen stopper ved hvert resultat inntil du trykker på A igjen.

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



Vent et halvt sekund

Ordet endres fra … til hei etter omtrent et halvt sekund. Skjermen fortsetter å tegne mens korrutinen er satt på pause.

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



Sirkler som ankommer én etter én

Fem sirkler drypper inn fra toppen, to bilder fra hverandre, og faller deretter.

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



To bilder, fra klokken

Sprite 1 og sprite 2 er to rammer av samme figur. Bildet bytter fire ganger i sekundet.

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

Hver funksjon

Argumenter i hakeparenteser er valgfrie. Indekser for tabeller og strenger starter på 1, unntatt DATA-rader og -kolonner, som starter på 0. Farger er palettindekser.

Indeks

`_tegne`
`_init`
`_mp`
`_mp_fail`
`_mp_bli med`
`_mp_forlate`
`_mp_oppdatering`
`_sett_fps`
`_oppdatering`

abs
achclear
achdef
oppnå
oppnådd
legge til
alle
hevde
atan2
band
bnot
bor
btn
btnd
btnp
btnu
bxor
kamera
tak
chr
circ
circfill
klikkl
klikkm
klikker
klippet
cls
samskape
farge
corsume
cos
costatus
telle
markøren
dcols
del
delikatesseforretning
delsave
dget
drukner
dset
dstr
fget
fyllp
snu
flr
foreach
fset
info
ipairs
taste av
tasten
keystat
keystr
linje
belastning
logg
lshr
kart
maks
memcpy
memset
mget
midten
min
mus
mset
musikk
musikkstat
kunngjøre
ncall
neste
nhost
bli med
nkb
nkbstr
nlan
nleave
nliste
nhør
merk
nspillere
nsend
nstun
ord
oval
ovalfyll
pakke
par
kompis
palrgb
palt
pcall
titt
titt2
titt4
perf
pget
stikke
stikke 2
stikke 4
skrive ut
print10
print8
pset
rekt
rectfill
rnd
rotl
rotr
rett
rettfyll
lagre
spareslot
skjermen
bla
velg
settkart
settpalett
setspr
settex
setttur
sfx

sfxget	synd	stat
sfxhdr	songget	sub
sfxset	sangsett	tid
sfxstat	splitt	tline
sget	spr	tonum
sgn	sqrt	tostr
shl	srand	pakke ut
showstats	sset	whoami
shr	sspr	utbytte

Livssyklus

_init

`_init() ... end`

Kjører én gang når spillkassetten starter. Kode som sitter utenfor en hvilken som helst hook foldes også inn i _init.

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

_update

`_update() ... end`

Spilllogikk. Kalles ved 60 Hz, selv når _draw er begrenset eller verten er tregere. Les knapper her.

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

_draw

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

Tegner én ramme. Uten argument, eller 0, kjører den så ofte som verten presenterer. Send n (1..999) for å begrense tegningen til n rammer per sekund. _update forblir på 60 Hz.

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

_set_fps

```
_set_fps(n)
```

Endrer _draw-grensen mens spillet kjører. 0 betyr ubegrenset. n er begrenset til 0..999.

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

flip

```
flip()
```

Presenterer skjermen og gir etter frem til neste bilde. Bruk den bare hvis du tegner din egen bildeløkke. Ikke vent med å holde deg opptatt etter den.

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

time

```
time()
```

Sekunder siden _init.

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

Inndata

btn

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

1 mens knapp b holdes inne, ellers 0. p er spilleren, 0..7. Uten argumenter returnerer den en bitmaske for spiller 0. Knapper: 0 venstre, 1 høyre, 2 opp, 3 ned, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 velg, 11 start.

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

btnd

`btnd(b, [p])`

Sann bare på _update-rammen når knapp b trykkes ned.

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

btnp

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

Sann ved trykk, deretter igjen mens den holdes inne: 30 tikk med forsinkelse, deretter hvert 8. tikk. Uten knapp returneres en maske.

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

btnu

`btnu(b, [p])`

Sant bare på rammen når knappen slippes.

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

mouse

`mouse()`

Returnerer musens x, y i spillpiksler.

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

clickl

`clickl([p])`

Venstre museknapp. clickl() holdes nede. clickl(1 er presskanten.

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

clickr

`clickr([p])`

Høyre museknapp. Samme kant med ett argument som ved klikk.

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

clickm

`clickm([p])`

Midtknapp, eller et klikk med rullehjulet.

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

scroll

`scroll()`

Musehjulbevegelse i denne rammen. Positiv og negativ avhenger av hjulretningen.

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

keyon

`keyon()`

Åpner et tekstfelt som styres av knapper. Opp og ned endrer bokstaven A–Å. A skriver den bokstaven. X sletter. Y er mellomrom. B bekrefter og lukker feltet. Tastaturet skriver ikke inn i det.

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

keyoff

`keyoff()`

Skjuler tekstfeltet.

```
keyoff()
```

keystr

`keystr([s])`

Tekst i feltet, deretter bekrefter 1 på ramme B, og så endres bokstavene opp og ned. Bokstaven er tom når feltet lukkes. keystr("") fjerner teksten.

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

keystat

```
keystat()
```

1 mens tekstfeltet er åpent, ellers 0.

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

Tegning

screen

```
screen()
```

Gjeldende skjermstørrelse for spillkassetten i piksler. Returnerer bredde, deretter høyde. Dette er spillets oppløsning, ikke redigeringsvinduet.

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

cls

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

Tømmer skjermen. Standardfargen er 0.

```
cls(1)
```

color

```
color(col)
```

Angir tegnefargen som brukes når et senere kall utelater farge. Uten argument returneres gjeldende farge.

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

camera

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

Offseter hvert tegneanrop. camera() tilbakestilles til 0, 0.

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

clip

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

Begrenser tegningen til et rektangel. Et femte argument forskjellig fra null skjærer det gjeldende klippet. clip() gjenoppretter fullskjermvisning. Peek av klipperegistrene er skrivebeskyttet.

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

cursor

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

Angir hvor neste utskrift uten koordinater skal gå, og eventuelt returnerer color.cursor() x, y.

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

pset

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

Angir én skjermpiksel. Teller mot pikselstatistikken.

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

pget

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

Fargen på skjermpikselen 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). Uten x og y skrives den ut ved markøren og flyttes til neste linje.

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

log

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

Legger argumentene, atskilt med mellomrom, til en tekstbuffer. pix37_run skriver ut den bufren ved stdout. Den tegner ikke.

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

print8

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

Samme som utskrift, med Fusion-skrifttypen 8 piksler.

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

print10

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

Samme som utskrift, med 10 pikslers Fusion-skrifttype.

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

line

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

Linje fra x0,y0 til x1,y1. To argumenter fortsetter fra forrige ende. line() tilbakestiller den kjeden.

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

circ

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

Sirkelomriss. Radius er i piksler.

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

circfill

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

Fylt sirkel.

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

oval

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

Ellipsekontur innenfor avgrensningsboksen. Hjørner er inkluderende.

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

ovalfill

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

Fylt ellipse i en avgrensningsboks.

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

rect

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

Rektangelformet omriss. Hjørner er inkludert.

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

rectfill

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

Fylt rektangel.

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

rrect

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

Avrundet rektangelomriss. x og y er øverst til venstre. w og h er størrelsen i piksler; begge må være større enn 0. r er hjørneradiusen og standardverdien er 0. Den er festet til halvparten av den korte siden.

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

rrectfill

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

Fylt avrundet rektangel. Samme argumenter som rrect.

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

fillp

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

16-bits fyllmønster for senere fylte former. 0 er heltrukket. fillp() returnerer gjeldende mønster. Bitene går i 4×4-mønsteret.

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

pal

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

Tilordner farge c0 til c1 på nytt. Modus 0 er tegnepaletten, 1 visningspaletten, 2 fyllmønsterpaletten. pal() tilbakestiller.

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

palrgb

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

Setter én palettoppføring til RGB, hver kanal 0..255.

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

palt

`palt([col, t])`

Gjennomsiktighet. palt(col) eller palt(col, true) gjør col gjennomsiktig. palt(col, false) gjør den heltrukket. palt() gjør alle farger ugjennomsiktige.

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

setpalette

`setpalette(n)`

Velger palettbank n. setpalette() returnerer gjeldende bank.

```
setpalette(1)
```

tline

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

Tegner en linje på skjermen mens den sampler kartpiksler fra kartposisjonen mx, my, mdx og mdy. Beveger karteksemplet langs linjen.

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

Sprites og kart

spr

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

Tegner sprite n ved x, y. w og h er størrelsen i piksler, standard 8. fx og fy vender horisontalt og vertikalt når de ikke er null. tex er sprite-arkets fane. rot er vinkelen i svinger (0..1), samme retning som sin og cos. 0 lar sprite være urotert.

```
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 av sprite-arket til skjermen, skalert til dw med dh. Standardverdiene beholder kildestørrelsen.

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

sget

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

Fargen på én tekstel på sprite-arket.

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

sset

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

Skriver én tekstel på sprite-arket.

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

fget

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

Sprite-flagg. fget(n) returnerer flaggbyten. fget(n, f) er sann når flaggbiten f (0..7) er satt.

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

fset

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

Setter sprite-flagg. fset(n, v) skriver hele byten. fset(n, f, v) setter én bit.

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

setspr

```
setspr(n)
```

Velger det aktive sprite-arket. setspr() returnerer gjeldende ark. Samme bank som settex.

```
setspr(1)
```

settex

```
settex(n)
```

Alias av setspr.

```
settex(0)
```

map

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

Tegner kartet. Celle cx, cy på kartet tegnes på skjerm sx, sy. cw, ch på 0 betyr hele kartet. Laget aksepteres og ignoreres. tex er flisesettet.

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

mget

`mget(cx, cy)`

Sprite-ID i kartcelle cx, cy.

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

mset

`mset(cx, cy, n)`

Skriver sprite n inn i kartcellen cx, cy.

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

setmap

`setmap(n)`

Velger kart n. setmap() returnerer gjeldende kart. Kart 0 er et reelt kart.

```
setmap(1)
```

Hukommelse

peek

`peek(addr)`

Leser én byte av 64 KB-vinduet. Vinduet er den levende skjermen, deretter sprite-piksler, og mapper deretter celler når skjermen får plass til 65280 byte. Større skjermer pagineres med poke(0x5F40, page). Adresser som ikke er mappet returnerer 0. Også skrevet @addr.

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

poke

`poke(addr, ...)`

Skriver byte til addr. Ekstra argumenter skriver følgende byte. Umappede adresser ignoreres. Skrives også $addr, val. Å dytte skjermen teller som pikselskriving.

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

peek2

`peek2(addr)`

Usignert 16-bits verdi, little-endian, ved adress.

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

poke2

`poke2(addr, val)`

Skriver en usignert 16-bits verdi, little-endian.

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

peek4

`peek4(addr)`

Usignert 32-bits verdi, little-endian.

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

poke4

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

Skriver en usignert 32-bits verdi, little-endian.

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

memcpy

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

Kopierer len bytes innenfor det samme 64 KB-vinduet.

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

memset

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

Fyller len-byte som starter ved dest med val.

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

Matematikk

abs

```
abs(x)
```

Absolutt verdi.

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

flr

```
flr(x)
```

Runder mot minus uendelighet.

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

ceil

```
ceil(x)
```

Runder mot pluss uendelighet.

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

max

```
max(a, b)
```

Størst av de to verdiene.

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

min

```
min(a, b)
```

Den minste av de to verdiene.

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

mid

```
mid(a, b, c)
```

Midt av de tre verdiene. Nyttig som klemme.

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

sgn

`sgn(x)`

Tegn på x: -1, 0 eller 1.

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

sqrt

`sqrt(x)`

Kvadratrot. Et negativt argument er ikke et tall.

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

sin

`sin(x)`

Sinus x går i sving: 0, 0,25, 0,5, 0,75 går én gang rundt sirkelen. Ikke radianer.

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

cos

`cos(x)`

Cosinus, igjen, samme konvensjon som sinus.

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

atan2

`atan2(dx, dy)`

Vinkelen til vektoren dx, dy, i svinger fra 0 til 1.

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

rnd

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

Tilfeldig tall 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)
```

Setter frø i tilfeldighetsgeneratoren slik at en senere rundesekvens kan gjentas.

```
srand(1)
```

band

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

Bitvis OG. Også skrevet a og 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 en << n.

shr

```
shr(a, n)
```

Aritmetisk forskyvning til høyre. Også skrevet som et >> n.

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

lshr

```
lshr(a, n)
```

Logisk forskyvning til høyre. Også skrevet som et >>> n.

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

rotl

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

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

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

rotr

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

Roter bitene til høyre. Også skrevet som >><< n.

Tabeller og verdier

add

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

Legger v til tabell t. Hvis i er gitt, settes inn ved den 1-baserte indeksen.

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

del

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

Fjerner den første oppføringen som er lik v.

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

deli

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

Fjerner indeks i og returnerer den verdien. Uten i, fjernes den siste oppføringen.

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

all

```
all(t)
```

Iterator over arraydelen, i rekkefølge. Hull stopper gangen slik en 1-basert array slutter.

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

foreach

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

Kaller fn(v) for hvert arrayelement.

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

pairs

```
pairs(t)
```

Iterator over hver nøkkel og verdi, inkludert navngitte felt.

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

ipairs

```
ipairs(t)
```

Iterator over arraydelen med den 1-baserte indeksen.

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

next

`next(t, [k])`

Returnerer neste nøkkel og verdi etter k. next(t) starter i begynnelsen. Brukes av par.

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

count

`count(t, [v])`

Lengden på arraydelen. Hvis v er gitt, teller den hvor mange oppføringer som er lik v.

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

pack

`pack(...)`

Pakker argumentene inn i en ny tabell.

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

unpack

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

Returnerer t[i] til t[j] som separate verdier. Standardverdiene dekker hele arraydelen.

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

select

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

select("#", ...) returnerer hvor mange argumenter som følger. select(i, ...) returnerer argumentene fra i og utover.

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

assert

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

Stopper med en feil når c er usann eller null. msg erstatter standardteksten.

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

pcall

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

Kall f. Ved suksess returnerer `true` og resultatene. Ved feil returnerer `false` og feilteksten.

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

Strenger

chr

```
chr(n)
```

Ett 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-baserte. Standardinnstillingen er det første tegnet.

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

sub

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

Delstreng fra i til j, 1-basert. Utelat j for å ta halen.

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

split

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

Deler s inn i en tabell. sep har komma som standard. Et tall sep er en chunkbredde. conv konverterer brikker med tonum når den er sann.

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

tonum

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

Analyserer et tall. Grunntall kan være 2..36. Returnerer null når teksten ikke er et tall.

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

tostr

`tostr(v, [digits])`

Gjør om en verdi til tekst. siffer angir desimaler for tall.

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

Lyd

sfx

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

Spiller av mønster n og returnerer kanalen. Utelat ch, eller pass på -1, og den første ledige kanalen velges. ch -2 stopper mønster n på hver kanal. n -1 stopper en kanal, eller hver kanal når ch utelates. n -2 lar et løkkemønster fullføres i stedet for å løkke. sfx() uten argumenter returnerer -1.

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

sfxget

`sfxget(n, step)`

Returnerer en tabell med ett mønstertrinn: [1] tonehøyde, [2] bølge, [3] volum, [4] effekt. Tonehøyden er 0,63 (C-0 til D#5). Bølgen er 0,15. Volum er 0,7, og 0 er stille. Effekten er 0,7.

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

sfxset

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

Skriver ett mønstertrinn. Alle seks argumentene er obligatoriske.

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

sfxhdr

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

Mønsteroverskrift. sfxhdr(n) returnerer overskriftstabellen. sfxhdr(n, key, val) setter ett felt: speed, loop_in, loop_out, level, noise, crush, dtune, rverb, dampn, swing, steps. Tallene 0..10 velger de samme feltene i den rekkefølgen. speed er TIC, ticks per step. Higher er tregere. 15 er 120 slag per minutt. 16-bit aksepterer 1..255. Smalere synther fester seg til TIC-listen sin.

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

sfxstat

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

Mønster-ID som spilles av 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 uten mønster. Tonehøyde 0..63, bølge 0..15, volum 0..7, effekt 0..7. ticks er hvor lenge den holder, i de samme 120 Hz-tickene som TIC. Utelat ticks, og den varer i 15, ett standardtrinn. Effekten går over den tiden, deretter blir kanalen stille.

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

music

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

Starter sang n fra toppen. music(-1) stopper. Hvis du kaller musikk med ID-en som allerede spiller, lar den kjøre. music() returnerer sang-ID-en som spilles, eller -1. fade er millisekunder: sangen fades inn, eller fades ut når n er -1. maskebiter som er satt er kanalene som brukes for sangen. 0 betyr alle åtte. De andre kanalene forblir ledige for sfx.

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

musicstat

```
musicstat()
```

Tabell over sangen som spilles. sang og rad er sang-ID-en og raden, eller -1 når ingenting spilles. step og pos har åtte tall. step[1] er kanal 0, step[8] er kanal 7. step er noteindeksen, eller -1 når den kanalen er stille. pos er hvor langt noten har gått, fra 0 til 1. tick er hvor mange ticks denne raden allerede har spilt. played er hvor mange rader som har startet siden music(), inkludert den som spilles.

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

songget

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

Mønster-ID lagret i en sangcelle. ch er 0..7. 255 betyr at cellen er tom. Dette er det aktive kallet; kodehintteksten viser fortsatt en eldre argumentrekkefølge.

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

songset

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

Skriver en mønster-ID inn i en sangcelle. Bruk 255 for å tømme cellen.

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

stat

```
stat(n)
```

Kun lyd. 16..23 er sfx-ID-en på kanal n-16. 24..31 er musikkmønsteret på kanal n-24. Enhver annen n returnerer 0. Bildetiming og RAM er perf og info, ikke stat.

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

Datatabeller

dget

`dget(t, row, col)`

Leser en celle i datatabellen t fra DATA-panelet. Rad og kolonne starter på 0.

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

dset

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

Skriver en celle. Bruk en streng for en tekstkolonne.

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

dstr

`dstr(t, row, col)`

Cellen som tekst. Tall konverteres.

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

drows

`drows(t)`

Hvor mange rader tabell t har.

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

dcols

```
dcols(t)
```

Hvor mange kolonner tabell t har.

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

Lagre og prestasjoner

save

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

Lagrer v under nøkkelen k i lagringsfilen. v kan være et tall, en streng eller en tabell over disse. Filen er komprimert.

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

load

```
load(k)
```

Leser nøkkel k. Returnerer null når nøkkelen mangler.

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

delsave

```
delsave(k)
```

Sletter nøkkel k fra lagringen.

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

saveslot

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

Velger lagringsfilen i forfatterens mappe. Uten navn er spilleautomaten spilltittelen. Returnerer 1 når filen allerede finnes.

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

achieve

```
achieve(i)
```

Låser opp prestasjon i, fra 0 til 15. Lagres lokalt. En Steam-bygg setter også ACH_i. En katalogøkt rapporterer det til spillerens konto.

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

achieved

```
achieved(i)
```

1 når prestasjon i allerede er opplåst.

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

achdef

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

Skåltekst for prestasjon i, vist i 15 sekunder fra bunnen av skjermen. spr er en valgfri 8×8-flis, tekst arket sitt.

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

achclear

```
achclear(i)
```

Fjerner prestasjon i i den lokale filen. En Steam ACH_i som allerede var satt, forblir satt.

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

Nettverk

whoami

```
whoami()
```

Spiller-ID-en din er 0..15 mens en økt pågår. Offline er den -1.

```
id = whoami()
```

nhost

```
nhost()
```

1 hvis denne maskinen er verten, ellers 0.

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

nplayers

```
nplayers()
```

Hvor mange live-ID-er finnes i 0..15.

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

nlisten

```
nlisten()
```

Starter å være vert for UDP og returnerer LAN-adressen som ip:port.

```
addr = nlisten()
```

njoin

```
njoin(addr)
```

Kobler til ip eller ip:port.

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

nleave

```
nleave()
```

Forlater økten. Motpartene ser _mp_leave med why 0.

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

nsend

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

Sender en tapsfylt melding. Navnet er hendelsen. Opptil fire ekstra verdier følger. p fra 0 til 15 hviskinger til den motparten; utelat p for å sende til alle.

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

ncall

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

Samme form som nsend, men meldingen prøves på nytt til den bekreftes. Tre feil på rad avslutter økten.

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

nlan

```
nlan()
```

Gjeldende LAN-ip:port etter nlisten.

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

nstun

```
nstun()
```

Din offentlige IP-port fra STUN. En gjest på et annet nettverk kan prøve å bli med.

```
pub = nstun()
```

setturn

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

TURN-reserve når en direkte sti feiler. setturn() med tomme argumenter slår den av. Adressen er host:port.

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

nlist

```
nlist(game)
```

Tabell over åpne økter for det spillnavnet. Hver oppføring har adresse, spillere og navn.

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

nannounce

```
nannounce(game)
```

Hjerteslag, så denne verten vises i nlist for spillet. Kall den på en timer mens den er vert.

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

nkb

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

Adressefelt. nkb(1) åpner det, nkb(0) skjuler det, nkb() forteller deg om det er åpent. Opp og ned endrer et tegn, 0–9 eller et punktum eller et kolon. A skriver det. X sletter. Tastaturet skriver ikke inn i det.

```
nkb(1)
```

nkbstr

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

Tekst i adressefeltet, deretter endres tegnene opp og ned. nkbstr("") fjerner teksten.

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

_mp

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

Du skriver denne funksjonen. Den kjører når en melding kommer inn. `from` er avsender-ID-en. `name` er hendelsen. Opptil fire verdier 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 funksjonen. En peer mottok id p. Den utløses også for deg; whoami() er den id-en.

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

_mp_leave

`_mp_leave(p, why)`

Du skriver denne funksjonen. Peer p til venstre. Hvorfor 0 er en normal leave. Hvorfor 1 er en tapt forbindelse.

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

_mp_update

`_mp_update()`

Du skriver denne funksjonen. Den kjører hver 15. _oppdateringsramme, omtrent 4 Hz. Et bra sted å sende posisjoner.

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

_mp_fail

`_mp_fail()`

Du skriver denne funksjonen. Økten er død: verten forsvant, eller ncall mislyktes tre ganger.

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

Koroutiner

cocreate

`cocreate(f)`

Lager en korutine fra funksjon f. Den kjører ikke før koresume.

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

coresume

`coresume(c, ...)`

Kjører coroutine c til den gir eller returnerer. Ved suksess: sann og de gitte verdiene. Ved feil: usann og feilen.

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

costatus

`costatus(c)`

«suspendert», «løpende» eller «død».

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

yield

`yield(...)`

Gir kontrollen tilbake til den som kalte opp kjerneverdien. Verdiene blir resultatene av kjerneverdien.

```
yield(1)
```

Måle

info

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

Tegner live HUD i _draw: fps (gjennomsnitt på 1 sekund), upd, draw, px, max, ram, peak, ramc og net når en økt er over. info() bruker utskriftsmarkøren. info(col) angir bare fargen. info(x, y) eller info(x, y, col) plasserer den. Tall bruker B, k eller m slik at de passer på en liten skjerm. Selve samtalen telles ikke i piksler.

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

showstats

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

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

```
showstats(1)
```

perf

```
perf(n)
```

Ett HUD-tall, slik at du kan tegne det selv. 0 fps, 1 oppdatering, 2 tegninger, 3 piksler, 4 maks (tyngste bilde dette sekundet), 5 ram, 6 ramc (ram pluss den pakkete spillkassetten), 7 maksimal ram siden spillkassetten startet.

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