PIX37 Handbuch

Erst bearbeiten, dann programmieren. Offline-Kopie des Buches unter pix37.party/docs.

Editor

PIX37 DEV ist der Editor. Hier zeichnen, schreiben und hören Sie das Spiel und exportieren anschließend eine Spielkassette oder einen eigenständigen Build. Er kann auch Spielkassetten abspielen und PIX37.PARTY öffnen, um Spielkassetten herunterzuladen und den Katalog zu nutzen. PIX37 DEV EDUCATIONAL ist derselbe Editor und kostenlos. Er bietet keine Builds, keinen Export von Spielkassetten und keinen Zugriff auf PIX37.PARTY. PIX37 Console ist der Player. Er spielt `.p37`-Spielkassetten ab und ermöglicht das Herunterladen von Spielkassetten und das Durchsuchen des Katalogs über PIX37.PARTY.

Projekte und Spielkassetten

Eine `.pix37`-Datei ist das Projekt: Quellcode-Tabs, Sprite-Sheets, Karten, Sound und Datentabellen. Eine `.p37`-Datei ist die Spielkassette, die der Spieler ausführt: dicht gepackter Bytecode.

Spielstände und Erfolge befinden sich im Autorenordner neben den App-Daten. Unter Windows heißt ein Spielstand beispielsweise `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s`, ein Erfolg `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach`. Unter Linux heißen die entsprechenden Dateien `~/.local/share/PIX37/Mira/Pong.p37s` und `~/.local/share/PIX37/Mira/Pong.ach`. „Ada“ ist der Benutzername auf dem Computer. „Mira“ ist der Name des Autors und des Ordners. „Pong“ ist der Speicherplatz, der standardmäßig den Spieltitel enthält. `save` schreibt die Datei `.p37s`. Der erste freigeschaltete Erfolg schreibt die Datei `.ach`. Sind die Dateien bereits vorhanden, liest der Runner sie beim Start der Spielkassette.

Entwicklermenü

Die Escape-Taste öffnet das Entwicklermenü. Während ein Spiel läuft, kehrt man durch Drücken der Escape-Taste zum Editor zurück. Bestätigen Sie mit der Eingabetaste oder der A-Taste (standardmäßig V). Zurück geht es mit der B-Taste (standardmäßig C).

Wenn kein Projekt geladen ist, lautet das Menü: PROJEKT ERSTELLEN, PROJEKT LADEN, BUILD, Spielkassette LADEN, VERBINDUNG MIT PIX37.PARTY, EINSTELLUNGEN.

Wenn ein Projekt geladen ist, lautet die Reihenfolge: PROJEKT SPEICHERN, PROJEKT LADEN, PROJEKT ERSTELLEN, ASSETS, BUILD, Spielkassette LADEN, VERBINDUNG MIT PIX37.PARTY, EINSTELLUNGEN.

Strg+S speichert das geöffnete Projekt. Strg+R führt es aus.

Ein Projekt erstellen

PROJEKT ERSTELLEN fragt nach der Spielkassette, bevor Sie mit der Projekterstellung beginnen können.

- **VIDEO** — 1-BIT (2 Farben), 2-BIT (4), 4-BIT (16), 8-BIT (256).
- **AUDIO** — Abtasttiefe des Synthesizers: 16, 8, 4, 2 oder 1 Bit.
- **Bildschirm** – Voreinstellungen umfassen 64×64, 128×128, 160×90, 256×256 und 320×180. Mit der Option „Benutzerdefinierte Größe“ können Sie jede beliebige Größe von 1×1 bis 512×512 auswählen.
- **PALETTEN** – Farbbibliotheken. Die erste Bibliothek enthält die unten aufgeführten 16 Farben. Mit „PALETTE LADEN“ wird eine integrierte Liste angewendet: PICO-8 16, PICO-8 32, Game Boy, 1-Bit, CGA, Sweetie-16 und eine längere Liste von Lospec (Resurrect 64, Apollo, Endesga 32 und 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century u. a.). Sie können auch eine Palettendatei laden oder Ihre eigene PNG-Datei laden und die Palette aus diesem Bild extrahieren.



Farbindizes, die von `cls`, `pset`, `spr` usw. verwendet werden. Sie können jeden Index im Paletteneditor neu zeichnen.

Die fünf Tafeln

Der Editor besteht aus SPIEL, MALEN, CODE, SOUND und DATEN (in dieser Reihenfolge). Die Karte befindet sich im Menü MALEN. Soundeffekte und Lieder befinden sich im Menü SOUND. Klicken Sie auf ein Panel, um darin zu arbeiten. Wenn das Menü CODE den Fokus hat, ignorieren SOUND und DATEN die Buchstabentasten, sodass beim Tippen nicht gleichzeitig Klavier gespielt wird.



Vollständiges Layout, geschrieben `1:1:4:1:1`. Jede Zahl gibt an, wie viel Platz das jeweilige Panel in der Zeile einnimmt: GAME, PAINT, CODE, SOUND, DATA. Ein kleiner Bildschirm befindet sich zentriert in einer 128×128 Pixel großen Zelle. Ein größerer Bildschirm verbreitert die Spalte GAME.

EINSTELLUNGEN → GLOBAL → LAYOUT schaltet die Anordnungen durch:

Code	Was Sie sehen
`1:1:4:1:1`	Alles. SPIEL und SOUND übereinander, PAINT und DATEN übereinander, CODE rechts.
`1:1:4:1:1 2X`	Die gleiche Anordnung. Der Code wird mit doppelter Dichte gezeichnet, sodass doppelt so viel Text Platz findet.
`1:0:1:0:1`	SPIEL, DATEN, CODE.
`1:0:1:0:1 2X`	Die gleiche Anordnung, jedoch mit dem dichteren CODE.
`0:1:1:1:1`	Keine Spielvorschau. MALEN, CODE, SOUND, DATEN.
`0:1:1:1:1 2X`	Die gleiche Anordnung, jedoch mit dem dichteren CODE.
`0:1:4:1:2`	Studio. Breiter CODE, ein höherer DATENStrip, PAINT und SOUND.
`0:1:4:1:2 2X`	Die gleiche Anordnung, jedoch mit dem dichteren CODE.

`0:0:4:0:4`	Nur CODE und DATEN, beides umfangreich.
`0:0:4:0:4 2X`	Die gleiche Anordnung, jedoch mit dem dichteren CODE.
`0:1:4:1:0`	Farbe, Klang und ein breiter Code.
`0:1:4:1:0 2X`	Die gleiche Anordnung, jedoch mit dem dichteren CODE.
`0:4:0:0:0`	Nur Farbe, 256×256, ein Pixel des Blattes entspricht einem Pixel der Leinwand.

Eine `2X`-Zeile behält dieselben Bedienfelder bei. Das CODE-Bedienfeld wird mit doppelter Auflösung gezeichnet und auf dieselbe Zelle skaliert, sodass Text, Tabulatoren und Spaltenabstand halb so hoch sind und doppelt so viel Quelltext auf den Bildschirm passt.

SPIEL

Die Vorschau des laufenden Spielmoduls. Klicken Sie auf das SPIEL-Panel oder drücken Sie Strg+R, um die Vorschau zu starten. Unter EINSTELLUNGEN → SPIEL-PANEL können Sie das Panel konfigurieren.

- **Vorschau** – Leerlauffarbe, kontinuierlich oder automatisches Neuladen beim Bearbeiten.
- **In einem neuen Fenster spielen** – das Spiel öffnet sich neben dem Editor.
- **Anzeige** – Vorschau als RGB888 oder RGB565.
- **GAME 64** — ein 64×64-Bildschirm wird entweder 1:1 in der Zelle zentriert oder 2× skaliert, um sie auszufüllen.
- **Shader** – CRT und Filter im Spiel, getrennt von den Filtern im Editor.
- **Farbe** – Hintergrund des Bedienfelds.

MALEN

Sprite-Sheets und, auf der Kartenseite desselben Panels, Kachelkarten. Die Sheets können größer als der Bildschirm sein. Die Sprite-Vorschau ist in den Größen 8, 16, 32, 64 und 128 Pixel verfügbar (128 Pixel nur, wenn das Sheet mindestens 128 Pixel groß ist). Bewegen Sie das Mausrad über die Sprite-Vorschau, um sie zu vergrößern oder zu verkleinern. Die Pinselgrößen sind 1, 2, 3 und 4.

Schlüssel	Werkzeug
B	Bleistift
M	Rechteck auswählen
Q	Lasso

S	Stempel
G	Eimer
ICH	Pipette
V	Bewegen
L / Umschalt+L	Linie / Kurve
U / Umschalt+U	Hohlrechteck / Hohlellipse
F / Umschalt+F	Ausgefülltes Rechteck / ausgefüllte Ellipse
D	Dither
[ ]	Kleinere / größere Bürste
Strg+Z / Strg+Y	Rückgängig / Wiederherstellen. Strg+Umschalt+Z stellt die Aktion ebenfalls wieder her.
Strg+C / Strg+X / Strg+V	Kopieren, Ausschneiden, Einfügen
Strg+A	Wählen Sie die Leinwand aus.

Die Karte verwendet dasselbe Prinzip, jedoch mit weniger Werkzeugen: B (Stift), M (Rechteck auswählen), S (Stempeln), V (Verschieben) und Strg+Z, Strg+Y, Strg+C, Strg+V. Halten Sie die Alt-Taste gedrückt, um die Kartentextur zu vergrößern oder zu verkleinern. Die Pinselgröße auf der Karte entspricht der Größe der Malwerkzeuge.

Jeder Sprite besitzt acht Flag-Bits. Paint setzt diese auf dem Sprite; `fget` und `fset` lesen sie im Spiel aus.

EINSTELLUNGEN → MALEREDITOR ist der Ort, an dem dieses Bedienfeld konfiguriert wird:

- **Palette** – welche Bank wird verwendet, plus Laden und Speichern.
- **Vorschau** — RGB888 oder RGB565.
- **Pinselcursor** – der Zeiger während des Malens.
- **Pixel-Snap** – wie der Pinsel an den Pixeln einrastet.
- **Großer Pinsel** – ein großer Pinsel schnappt von seiner oberen linken Ecke oder von seiner gesamten Grundfläche ab.
- **Zwiebelhaut** – ein Schatten anderer Bilder und in welche Richtung dieser Schatten blickt.
- **Kartenzoom** – echte Pixel oder freier Zoom.
- **Farben** – die Farben der Benutzeroberfläche.

CODE

Tabs sind separate Dateien innerhalb des Projekts. Klicken Sie auf einen Tab-Namen, um ihn umzubenennen; mit Enter bestätigen Sie. Mehrere Tabs, die jeweils `_init`, `_update` oder `_draw` definieren, werden in Tab-Reihenfolge zusammengefügt: Der erste Tab wird zuerst ausgeführt, der letzte zuletzt. Code, der sich nicht innerhalb eines Hooks befindet, wird Teil von `_init`. Eine Spielkassette ohne Hooks wird als eine `_draw`-Datei behandelt.

Schlüssel	Aktion
Pfeile	Bewegen Sie den Cursor. Mit der Umschalttaste wird die Auswahl erweitert.
Strg+Links/Rechts	Per Wort.
Pos1 / Ende, Strg+Pos1 / Ende	Anfang oder Ende der Zeile oder des Tabulators.
Tab	Fügen Sie zwei Leerzeichen ein. Akzeptieren Sie den Namen in der Vervollständigungsliste.
Eingeben	Neue Zeile, eingerückt wie die vorherige. Akzeptiert eine Vervollständigung.
Strg+C / X / V	Kopieren, ausschneiden, einfügen.
Strg+Z / Strg+Y	Rückgängig machen / wiederherstellen.
Strg+A	Wählen Sie die Registerkarte aus.
Strg+D	Duplizieren Sie die Zeile oder die Auswahl nach unten.
Strg+/	Kommentieren oder entfernen Sie die Kommentarzeichen mit `--`.
Strg+F	Suchen. F3 springt zum nächsten Treffer. Escape schließt die Leiste.
Strg+G	Gehe zu einer Zeilennummer.
Flucht	Schließt zuerst die Suchfunktion, die Gehe-zu-Funktion oder die Hinweisliste. Ein zweites Drücken der Escape-Taste führt vom Editor zum Entwicklermenü.

Während Sie einen Funktionsnamen eingeben, werden Hinweise angezeigt. Mit der Eingabetaste oder der Tabulatortaste wird der Name eingefügt, und falls die Option „VOLLSTÄNDIG" in den Einstellungen aktiviert ist, wird ein `(` angezeigt ( ). EINSTELLUNGEN → CODE-EDITOR:

- **Suche** — Strg+F im aktiven Tab oder in allen Tabs.
- **Text hoch** — Glyphen auf 5 px zuschneiden oder die vollständige Glyphe anzeigen.
- **Stringgröße** – Schriftgröße von Zeichenkettenliteralen.
- **Schriftart für Kommentare** – 3×5 oder die Fusion-Schriftart der aktuellen UI-Sprache.
- **Vervollständigen (** — ob die Vervollständigung die Klammern ausgibt.
- **Codehinweise** – ein- oder ausschalten.
- **Hinweisesprache** – die Menüsprache oder Englisch.

- **Caret-Space** – zusätzlicher Versatz in leeren Zeilen.
- **Rinne** — Zeilennummern.
- **Farbe** – Hintergrund, Kommentare, Tabulatoren, Zahlen, Operatoren, Schlüsselwörter, Zeichenketten, Engine-Funktionen, Ihre Funktionen, Variablen, Klammern, Caret-Zeichen, Auswahl, Fehlerzeile, Warnung.

KLANG

Das Bedienfeld hat zwei Registerkarten. „SFX" ist ein Pattern: eine kurze Notenfolge. „SOUND" ist ein Song: Hier werden die Patterns auf acht Kanäle verteilt. „TIC" gibt an, wie viele Ticks ein Schritt verstreichen lässt. Der Takt beträgt 120 Ticks pro Sekunde, ein Schritt entspricht einer Sechzehntelnote, also sind 15 Ticks 120 Schläge pro Minute. Ein höherer TIC-Wert bedeutet ein langsameres Pattern. Zwei Pattern bleiben im Takt, wenn sich ihre TIC-Werte teilen: 15 und 30 sind im Takt, 15 und 8 nicht.

Das Klavier spielt, sobald eine Note mit der Maus ausgewählt wird. Die weißen Tasten haben die Bezeichnung ZXCVBNM (CDEFGAB). Die schwarzen Tasten haben die Bezeichnung SDGHJ (C# D# F# G# A#). Bewegen Sie den Mauszeiger über einen beliebigen Wert und geben Sie eine Zahl ein oder bewegen Sie den Mauszeiger darüber und scrollen Sie mit dem Mausrad. Die Tonhöhenwerte im Code reichen von 0 (C-0) bis 63 (D#5).

Bei einem Pattern sind LOOP und LEN identisch. Steht OUT über 0, wird eine Schleife von IN bis OUT abgespielt, und die Bezeichnung lautet LOOP. Steht OUT bei 0 und IN über 0, entspricht dies LEN: So viele Noten werden vom Anfang an gespielt, dann endet das Pattern, und die Bezeichnung lautet LEN. Stehen beide bei 0, wird in jedem Schritt gespielt.

Wie viel vom Synthesizer Sie bearbeiten können, hängt von der für die Spielkassette gewählten Bittiefe ab. Unter EINSTELLUNGEN → SFX-EDITOR legen Sie diese Bittiefe fest, ob eine Note automatisch weitergeschaltet wird (aus, Notenkette oder Feldkette), ob die Leertaste die Wiedergabe neu startet oder Start und Stopp umschaltet, und die Farben des Editors.

16-Bit

Jeder Schritt hat eine Note, eine Lautstärke, eine Wellenform und einen Effekt. Die Lautstärke reicht von 0 bis 7. 0 bedeutet Stille.

Wellen:

- **0 Dreieck**
- **1 geneigte Säge**
- **2 Sägen**
- **3 Quadrate**
- **4-Puls** – ein schmaler Puls
- **5 Organ**
- **6 Geräusche**

- **7-Phaser**
- **8 PWM** – ein Quadrat, dessen Breite sich ändert
- **9 Impulse 25%**
- **10 sin**
- **11 metallisches Geräusch**
- **12 FM Glocke**
- **13 Zupfen**
- **14 Supersäge**
- **15 PWM (C64)**

Effekte auf eine einzelne Note:

- **0 keine**
- **1 Slide** – Tonhöhe und Lautstärke gleiten über diesen Schritt zur nächsten Note.
- **2. Vibrato** – Tonhöhenschwankungen
- **3 Drop** – die Tonhöhe sinkt bis zur Stille, der übliche Kick
- **4. Einblenden**
- **5 Ausblenden**
- **6-Arpeggio** – spielt diese Note und die nächsten drei, jeweils eine alle 8 Takte. Bei TIC 8 oder darunter alle 4 Takte.
- **7 schnelle Arpeggien** – die gleichen vier Noten, alle 4 Takte, oder alle 2, wenn der TIC 8 oder niedriger ist.

Die Musterüberschrift, oberhalb der Notizen:

- **TIC** – Ticks pro Schritt, von 1 bis 255. 15 entspricht 120 Schlägen pro Minute. Geben Sie die Zahl ein oder erhöhen Sie sie um eins.
- **LOOP IN / OUT** – Wenn OUT größer als 0 ist, wird das Muster von IN nach OUT wiederholt. Wenn OUT auf 0 gesetzt ist, lautet die Bezeichnung LEN: IN gibt die Anzahl der gespielten Noten an, und 0 bedeutet das gesamte Muster.
- **RAUSCH** — das gesamte Muster wird zu Rauschen.
- **CRUSH** – eine aufregende Variante jeder Welle.
- **DTUNE** — 0 aus, 1 ein leichter Chorus, 2 ein starker Chorus.
- **RVERB** — Hall: 0 aus, 1 kurz, 2 lang.
- **DAMPN** — ein Tiefpassfilter: 0 aus, 1 leicht, 2 stark.
- **LEVEL** — Lautstärke des gesamten Musters, 0 bis 7.
- **SCHWINGEN** – Schlurfen. Ungerade Schritte etwas länger warten, gerade Schritte etwas kürzer.
- **SCHRITTE** — Anzahl der Schritte des Musters: 32, 24, 16 oder 12.

8-Bit

Ein Schritt besteht aus einer Note und einer Wellenform. Die vier Wellenformen sind Rechteck, Dreieck, Sägezahn und Rauschen. Die Kopfzeile enthält die Felder TIC, LOOP IN/OUT, FX und LEVEL. TIC ist einer der

Werte 1 bis 60. FX ist ein Effekt für das gesamte Pattern: 0 = kein Effekt, 1 = Vibrato, 2 = Arpeggio, 3 = Ausblenden. LEVEL reicht von 0 bis 3.

4-Bit

Ein Schritt ist nur eine Note. Die Kopfzeile enthält TIC, LOOP (ein/aus), WAVE (die gleichen vier Wellenformen) und OCT (die Oktave, in der die Noten liegen). TIC ist einer der Werte 4, 6, 8, 10, 12, 15, 20 oder 30.

2-Bit

Eine Stufe kann leer, eine Trommel, ein Schlag oder ein Quetschvorgang sein. Der Kopf hat TIC und BASE. TIC ist 8, 15, 16 oder 30. BASE bestimmt die Höhe von Trommel und Schlag.

1-Bit

Ein Schritt entspricht einer Note. TIC ist 8, 15, 16 oder 30, die gleichen vier Geschwindigkeiten wie bei 2-Bit.

Lieder

Der Reiter „SOUND“ zeigt den geöffneten Song an. Acht Spalten repräsentieren die Kanäle. Jede Zelle enthält eine Muster-ID. Die leere Zahl 255 bedeutet, dass der Kanal in dieser Zeile stummgeschaltet ist. Die Spalte „X“ enthält einen Befehl: Fortsetzen, Stopp, Schleifenstart oder Schleifenende. „Stopp“ beendet den Song nach der aktuellen Zeile. „Schleifenende“ springt zurück zum letzten Schleifenstart.

Eine Reihe endet, sobald das am weitesten links stehende Muster mit „LOOP OUT“ bei 0 beendet ist. Weiter rechts liegende Muster verlängern die Reihe nicht: Ein längeres Muster wird abgebrochen, ein kürzeres beginnt von Neuem und endet so lange, bis die Reihe abgeschlossen ist. Ein Muster mit „LOOP OUT“ oberhalb von 0 bildet eine Schlaufe zwischen „LOOP IN“ und „LOOP OUT“ und beendet die Reihe nicht. Wenn alle Muster der Reihe eine Schlaufe bilden, bleibt die Reihe an dieser Stelle.

Abkürzungen

Schlüssel	Aktion
Klaviertasten	Schreiben Sie eine Notiz zu dem mit der Maus ausgewählten Schritt.
Aufwärts / Abwärts	Ändern Sie den Wert unter dem Cursor.
Raum	Spiele den geöffneten Soundeffekt oder den geöffneten Song aus der ersten Zeile ab. Starte/stoppe oder starte neu über die Einstellungen des Soundeffekt-Editors.
Strg+Z / Strg+Y	Rückgängig machen / wiederherstellen.

Strg+C / Strg+X / Strg+V	Auswahl kopieren, ausschneiden, einfügen.

DATEN

Benannte Tabellen. Jede Tabelle besteht aus Zeilen und Spalten. Die Codebeispiele `dget`, `dset`, `dstr`, `drows` und `dcols` verwenden diese Tabellen. Die Zeilen- und Spaltennummern im Code beginnen bei 0.

Spaltentypen, die im Header durchlaufen werden:

- **bool** — ein Bit.
- **2** — zwei Bits.
- **4** — vier Bits.
- **8** — ein Byte.
- **16** — 16 Bit, für IDs und größere Zahlen.
- **spr** — eine Sprite-ID.
- **str** — Text, dargestellt mit der Systemschriftart.
- **str8** und **str10** – derselbe Text, nur höher gezeichnet.
- **f16** und **f32** – eine Gleitkommazahl mit halber Genauigkeit und eine Gleitkommazahl mit voller Genauigkeit.

Wenn sich der Mauszeiger über einer Spaltenüberschrift befindet, ändert Strg+Scrollen den Spaltentyp. Ein einfaches Scrollen bewegt die Maus spaltenweise oder zeilenweise nach unten, je nachdem, was unter EINSTELLUNGEN → DATENEDITOR ausgewählt ist. Strg kehrt diese Richtung um, weshalb Strg+Scrollen über einer Überschrift den Typ ändert, wenn das Scrollrad auf Spalten eingestellt ist.

Mit den Pfeiltasten navigieren Sie. Mit der Eingabetaste bearbeiten Sie eine Zelle. EINSTELLUNGEN → DATENEDITOR:

- **Scrollrad** – Spalten oder Zeilen scrollen.
- **Farben** – das Raster, die Kopfzeile und die Auswahl.

Vermögenswerte

ASSETS importiert und exportiert das jeweils geöffnete Element, nicht das gesamte Projekt. Sprites verwenden den in PAINT geöffneten Sprite auf dem geöffneten Blatt. Die Karte ist die geöffnete Karte. SFX ist das geöffnete Muster. Musik ist das geöffnete Lied. Daten sind die geöffnete Tabelle. Code bildet die Ausnahme: Jeder Tab wird geschrieben und jeder Tab wird wieder gelesen.

Art	Datei	Was bewegt sich
Sprites	`.png`	Der offene Sprite.

Karte	`.p37map`	Die offene Karte.
SFX	`.p37sfx`	Das offene Muster.
Musik	`.p37mus`	Der Titelsong. Die Pattern-Nummern bleiben unverändert. Exportieren Sie die Soundeffekte separat, falls Sie diese benötigen.
Daten	`.p37dat`	Der offene Tisch.
Code	`.lua`	Jeder Tab.

Einstellungen

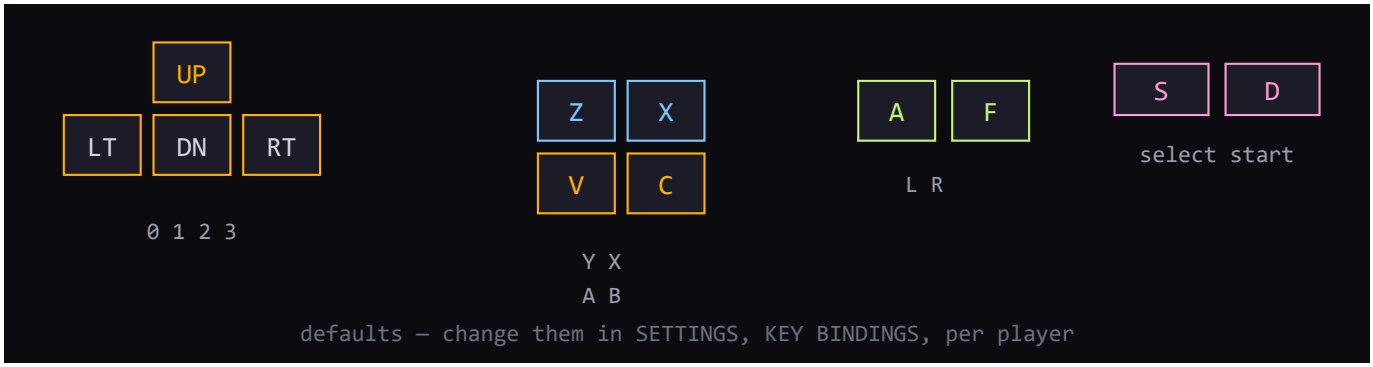
Die Wurzelliste:

Seite	Was es speichert
PROFILE	Alle Einstellungen speichern und laden. Neu, umbenennen, überschreiben, löschen. Profil als Text kopieren oder einfügen.
Speisekarte	Fenster-, Tasten- und Menüfarben, optionaler Hintergrund.
SPIELPANEL	Vorschau und Wiedergabefenster.
CODE-EDITOR	Hervorhebungen und Hinweise.
MALEREI-EDITOR	Pinsel, Palette, Kartenansicht.
SFX-Editor	Synthesizer und Noteneingabe.
DATENREDAKTEUR	Radrichtung und Farben.
SPRACHE	Menü- und Hinweissprache.
LIZENZEN	Lizenztexte, einschließlich der Schriftart Fusion Pixel und SDL3.

GLOBAL:

- **LAYOUT** — Anordnung der Bedienfelder, siehe oben.
- **LAUTSTÄRKE** — Master-Pegel, 0 bis 10.
- **TASTENBEZEICHNUNGEN** – Tasten für jeden der 8 Spieler. Drücken Sie eine Taste, um sie zu erfassen. Anweisungen: ABXYLR, Auswählen.
- **Präsens** – Leerlauffframes überspringen oder jeden Frame zeichnen.
- **Vollbild** – randlos.
- **Aktualisieren** – Fenster-FPS. Links und rechts in Schritten von 60, 90, 120 und maximal. Geben Sie mit der Eingabetaste eine Zahl bis 1000 ein. 0 bedeutet keine Begrenzung.
- **Vsync** – Warten auf die Anzeige zwischen den Frames.

- **Skalierung** – ganzzahliger Fenstermultiplikator.
- **Shader** – CRT und Filter im Editor, nicht im Spiel.
- **Farbe** – Menütext und Hervorhebung.
- **Hintergrund** – eine optionale PNG-Datei, zugeschnitten auf 128 Pixel, hinter dem Menü. Durch Klicken auf „Löschen" wird sie entfernt.
- **Editor-Cursor** – Zeigerform.
- **Maus** – schwebender Zeiger oder an Pixeln ausgerichtet.
- **UI-Geschwindigkeit** – wie schnell eine gedrückte Richtung in einer Liste wiederholt wird.
- **Explorer** – Dateilisten beginnen im Stammverzeichnis des Computers oder im Ordner der .exe-Datei.



Standardtastatur. A dient in Menüs zur Bestätigung. B ist die Zurück-Taste. Im Spiel sind dies die Tasten 4 und 5. Die Auswahltaste (S) ist Taste 10 und keine Menü-Tastenkombination.

Taste	Rolle	Standardschlüssel
0	Links	Pfeil nach links
1	Rechts	Pfeil nach rechts
2	Hoch	Pfeil nach oben
3	Runter	Pfeil nach unten
4	A, bestätigen	V
5	B, zurück	C
6	X	X
7	Y	Z
8	R	F
9	L	A
10	Wählen	S

11	Start	D

Bauen

Wählen Sie eine Plattform: Windows, Linux, Mac, Android, iOS oder Web.

- **Name** — Dateiname des Builds.
- **Autor** — unter Windows und Linux. Dies ist der Ordnername für die Speicherstände.
- **Steam-App-ID** – unter Windows und Linux. Eine leere ID bedeutet, dass Steamworks nicht installiert ist. Eine ausgefüllte ID ordnet `achieve(0)`…`achieve(15)` den Steam-Errungenschaften ACH_0…ACH_15 zu.
- **Id** – Bundle-Identifikator, nach dem die Plattform fragt.
- **Code ausblenden** – auf Desktop- und Mobil-Overlays. Dadurch wird die `peek`-Version der Spielkassette verlangsamt. Es handelt sich nicht um eine Sperre.
- **Vollbild** – der exportierte Player startet im Vollbildmodus.
- **Shader** – bis zu drei Ebenen, die in den Player integriert sind.
- **Download .p37** — Nur im Web. Zeigt einen Download-Eintrag im Pausenmenü an.
- **Symbol** – eine Kachel aus einem Sprite-Sheet: Sheet, Größe, x, y. Oder ein zugeschnittenes PNG-Cover.
- **Exportieren** – Ordner auswählen. Das Menü meldet „BUILD SAVED“ oder „BUILD FAILED“.

Erstellen Sie einen Build mit aktivierter Internetverbindung. Die Startseite fragt den Server möglicherweise anonym ab, ob der Hash dieses Builds bekannt ist. Diese Prüfung dient zwei Zwecken: Sie schützt mich, den Autor von PIX37, vor Kopien des Tools, die ich nicht weitergeben darf. Außerdem schützt sie den Entwickler des Spiels. Es kommt immer häufiger vor, dass fremde Spiele auf Plattformen wie Play (Android) oder iOS als eigene angeboten werden. Das Spiel startet trotzdem. Ist das Gerät offline, zeigt die Startseite an, dass keine Verbindung besteht. Diese Meldung dient dem Spieler als Hinweis darauf, dass die Kopie möglicherweise nicht aus einer legalen Quelle stammt. Es handelt sich um eine kleine Sicherheitsmaßnahme, damit Sie, falls Ihr Spiel ohne Ihre Zustimmung gestohlen und verkauft wurde, dies auf Plattformen wie Play (Android), Steam oder itch.io melden können. Der Server speichert den Zeitpunkt der ersten Hash-Registrierung neben dem Hash und ändert ihn später nicht mehr. Der frühere Zeitpunkt entspricht dem Original-Build. Ein später registrierter Hash kennzeichnet eine spätere Kopie.

Ich möchte mit dieser Überprüfung die Spieleentwickler schützen. Die Engine nimmt ihnen bereits viel Arbeit ab, und dies ist ein weiterer Aspekt. Mir ist bewusst, dass dies kontrovers diskutiert werden kann. Sollte die Community beschließen, auf diesen Mikro-DRM zu verzichten, werde ich dies tun.

Eine alte Spielkassette aus einem früheren Editor lässt sich nach dem Sprachwechsel möglicherweise nicht mehr starten. Öffnen Sie das Projekt `.pix37` in diesem Editor und exportieren Sie ein neues Projekt `.p37`.

Verbinde dich mit PIX37.PARTY

Die Verbindung zu PIX37.PARTY erfolgt innerhalb der App. Es gibt keine Anmeldeseite im Browser. Die Anmeldung erfolgt per Steam-Session-Ticket oder über itch.io mit Controller und Maus. Der Dienst zeigt einen eindeutigen Anzeigenamen mit 2 bis 16 Zeichen an. Lizenzschlüssel und Steam-ID werden anderen Spielern nicht angezeigt.

Nach den Nutzungsbedingungen umfasst der Hub die Bereiche Favoriten, Katalog, Rangliste, Spieler und Ihr Profil.

Eine Entwicklerlizenz berechtigt den Herausgeber. Du erstellst das Spiel, lädst es hoch und vergibst Sterne. Eine Spielerlizenz berechtigt den Spieler. Herzen stammen von Spielern, die auf einem Runner spielen.

- **Katalog** – Spiele durchsuchen. Filteroptionen umfassen Genre (bis zu drei), Bildschirmgröße, Größe der Spielkassette und ob der Upload ein `.pix37`-Projekt enthält.
- **Favoriten** – Spiele, die Sie markiert haben.
- **Upload** – Entwicklerlizenz erforderlich. Sie senden eine `.p37`-Datei und können das `.pix37`-Projekt anhängen. Eine `.p37`-Datei ist gepackter Bytecode. Dieser ist schwer lesbar und kann absichtlich verschleiert werden. Eine `.pix37`-Datei ist die eigentliche Projektdatei, mit der Sie arbeiten: Quellcode, Bilder und Sound in einem Format, das andere öffnen und analysieren können. Durch das Senden dieser Datei können andere Entwickler das Spiel selbst untersuchen, anstatt nur eine komprimierte Spielkassette. Titel, Autor, Genre, Cover, Symbol und Namen der Erfolge werden mitgesendet. Die übermittelte RAM-Nummer entspricht der maximalen Auslastung der Spielkassette.
- **Kommentare und Erfolge** – auf der Spielseite. Wenn Sie einen Erfolg freischalten, während Sie angemeldet sind, wird dieser sowohl auf Ihrem Konto als auch auf Ihrem Gerät gespeichert.
- **Rangliste** – Entwickler, sortiert nach Sternen oder Herzen. Jede Zeile zeigt beide Werte an. Klicken Sie auf einen Namen, um die Spiele dieses Entwicklers, sortiert nach Herzen, anzuzeigen.
- **Spieler** – Accounts sortiert nach freigeschalteten Erfolgen.

Code

Die Sprache ist ein kleines Lua-ähnliches Skript. Dieser erste Teil beginnt mit einem leeren Projekt, erklärt die Sprache und zeigt anschließend ein kurzes Beispiel für jede Funktion.

Ein erstes Programm

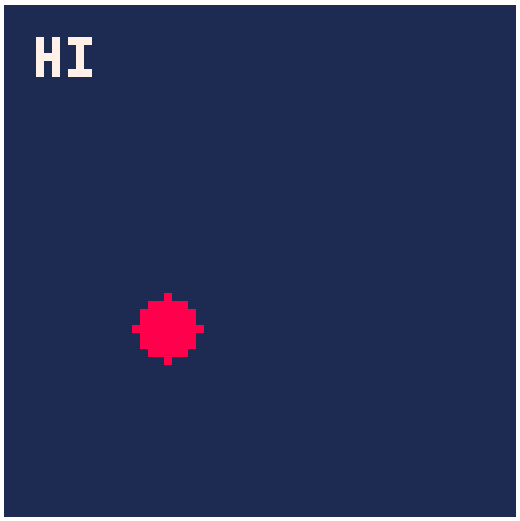
Erstelle ein Projekt mit 64×64 Pixeln und 4-Bit-Farbtiefe. Öffne CODE. Ersetze den Tab durch:

```
_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
```

Strg+R startet das Spiel. Mit den Pfeiltasten wird der Kreis bewegt. Mit Escape wird das Spiel beendet und man kehrt zum Editor zurück.



Die gleichen drei Befehle: `cls(1)` malt den marineblauen Bildschirm, `print` zeichnet HI und `circfill` zeichnet den rosa Kreis.

Was läuft und wann

- `_init` wird einmal ausgeführt.
- `_update` läuft mit 60 Hz. Hier können Sie Bewegungssensoren und Tasten anbringen.
- `_draw` rendert. Ohne FPS-Argument rendert es so schnell, wie es Computer und Monitor zulassen. `_draw(30)` begrenzt die Rendergeschwindigkeit auf 30 FPS und lässt die Logik bei 60 FPS.

- `_set_fps(n)` ändert diese Kappe später. 0 löscht sie.

Namen werden nicht nach Groß- und Kleinschreibung unterschieden; der Editor speichert sie in Kleinbuchstaben. `function` und `fun` bezeichnen dasselbe Schlüsselwort.

Sprache

Grundlagen der Programmierung

Ein Programm ist eine Liste der Zeilen, die das Spielmodul ausführt. Jede Überschrift zeigt zunächst die Struktur der Zeile, dann ein kleines Beispiel. Eine Zeile, die mit `--` beginnt, ist ein Kommentar: ein Hinweis für Sie. Das Spiel überspringt diese Zeile.

Variablen

Eine Variable ist ein Name, der einen Wert speichert. Sie wählen den Namen. Spätere Zeilen können ihn lesen und unter demselben Namen einen neuen Wert speichern.

Die Form ist `name = value`. `local name = value` sorgt dafür, dass dieser Name nur im aktuellen Block angezeigt wird, sodass er für den Rest der Spielkassette nicht sichtbar ist. Ohne `local` ist der Name global und für alle Tabs sichtbar.

```
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` bedeutet, dass kein Wert vorhanden ist. `true` und `false` bedeuten „Ja“ bzw. „Nein“. Ein Name, den Sie noch nie festgelegt haben, wird als `nil` angezeigt.

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

Tabellen

Eine Tabelle speichert mehrere Werte unter einem Namen. Eine Liste verwendet Zahlen. Das erste Element ist 1, nicht 0. Ein Datensatz verwendet Namen. Eine Tabelle kann eine andere Tabelle enthalten.

Listenform: `{ value, value }`. Datensatzform: `{ name = value, name = value }`. Eine Liste von Datensätzen ist eine Tabelle, deren Elemente Tabellen sind.

```
-- 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
```

Zustand und Körper

Eine Bedingung ist eine Frage mit Ja oder Nein als Antwort, z. B. `hp > 0` oder `btnd(4)`. Der Schleifenkörper ist der Codeabschnitt, der beim Betreten eines Blocks ausgeführt wird. `if` und jede Schleife haben dieselbe Struktur: eine Bedingung, dann ein Schleifenkörper, dann `end`. `repeat` schließt mit `until` anstelle von `end`.

Wenn

`if` wird nur ausgeführt, wenn die Bedingung erfüllt ist. `elseif` prüft eine andere Bedingung. `else` wird ausgeführt, wenn alle Bedingungen nicht erfüllt sind.

Form: `if condition then`, dann der Körper, dann `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
```

Schleifen

Eine Schleife wiederholt ihren Anweisungskörper. `while` prüft zuerst die Bedingung und wiederholt die Schleife, solange sie erfüllt ist. `repeat` führt den Anweisungskörper aus und prüft anschließend erneut, sodass der Anweisungskörper immer mindestens einmal ausgeführt wird. `for` zählt. Die letzte Zahl wird mitgezählt. Ohne eine dritte Zahl beträgt die Schrittweite 1. Liegt der Startpunkt also bereits nach dem Ende, wird der Anweisungskörper nicht ausgeführt. Für einen Rückwärtszähler ist eine negative Schrittweite erforderlich.

```
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
```

Gehen Sie an einem Tisch

`all` durchläuft jedes Element einer Liste, beginnend mit dem ersten. `pairs` durchläuft jeden Namen und den unter diesem Namen gespeicherten Wert.

```
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
```

Funktionen

Eine Funktion ist ein benanntes Rezept. Ihre Struktur ist `function name(arguments)`, gefolgt vom Funktionskörper und dann `end`. Sie wird durch Angabe des Namens und der Klammern aufgerufen. `return` gibt einen Wert zurück, und es können auch mehrere zurückgegeben werden. Die Argumente sind lokale Variablen der 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` ist eine Funktion, die auf die umgebenden lokalen Variablen zugreifen kann. Es handelt sich um eine Closure. Eine Tabelle kann eine solche Closure enthalten, und sie wird mit einem Punkt aufgerufen.

```
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, nein und die Wahl

`not` kehrt eine Bedingung um. `and` ist nur dann wahr, wenn beide Seiten wahr sind. `or` ist wahr, wenn mindestens eine Seite wahr ist. `a and b or c` wählt `b`, wenn `a` wahr ist, andernfalls `c`. `cond ? a : b` funktioniert genauso in einer Zeile: `a`, wenn die Bedingung wahr ist, und `b`, wenn sie falsch ist.

```
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)
```

Mathematik und Text

`+ - * /` steht für Addition, Subtraktion, Multiplikation und Division. `//` dividiert und behält die ganze Zahl. `%` gibt den Rest an. `^` ist die Potenz. `..` verbindet zwei Texte. `#t` gibt die Anzahl der Elemente in einer Liste an. Vergleiche sind `==` (gleich), `~=` oder `!=` (unterschiedliche) und `< > <= >=`. `x += 1` addiert 1 zu x. Die gleiche Form funktioniert für `-= *= /= ^= %=`.

```
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
```

Wo ein Einheimischer lebt

`local` bleibt so lange aktiv, bis `end` seinen Block schließt. `if` und `end` bilden jeweils einen eigenen Block. Sie können oberhalb der Zeile, die den Wert setzt, eine lokale Variable angeben, aber bis diese Zeile ausgeführt wird, ist der Wert `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
```

Springen

`::label::` markiert eine Zeile. `goto label` springt dorthin, jedoch nur innerhalb derselben Funktion.

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

Gedächtnis, kurz gesagt

`peek` und `poke` greifen über eine 16-Bit-Adresse auf Live-Pixel zu. Sie speichern keine zweite Kopie der Spielkassette. Die Adressen `0x5F00` bis `0x5FFF` sind Register für: Kamera, Clip (schreibgeschützt), Zeichenfarbe, Füllmuster, die drei Farbpaletten-Remaps und die Seitenzahlen. Die 65280 Bytes beidseitig dieser Öffnung bilden das Pixelfenster.

Wenn der Bildschirm in die 65280 Bytes passt (64×64 und 128×128 passen, 256×256 jedoch nicht), beginnt die Anzeige bei Adresse 0 und wird zeilenweise ausgeführt. Übrig bleibt das von `0x5F41` ausgewählte Sprite-Sheet (0 entspricht Sheet 0). Ist dieses Sheet größer als der verbleibende Speicherplatz, wählt `0x5F42` die Seite aus, und die Karte befindet sich nicht im Fenster. Passt das Sheet, besteht der Rest aus Kartenzellen der von `0x5F43` ausgewählten Karte (jeweils ein oder zwei Bytes). Passt die Karte nicht in den Rest, verteilt `0x5F44` die Karte auf mehrere Seiten.

Wenn der Bildschirm nicht passt, entspricht das gesamte Fenster einer Seite mit 65280 Pixeln. `poke(0x5F40, n)` blättert die Bildschirmseite. `poke(0x5F41, n)` mit n ungleich 0 zeigt stattdessen Blatt n an, und Blatt 0 ist in diesem Modus nicht sichtbar. `poke(0x5F43, n)` mit n ungleich 0 zeigt Karte n an, die Vorrang vor dem Blatt hat. Karte 0 ist in diesem Modus nur mit `mget` und `mset` verfügbar.

Ein `peek`, der nicht auf einem Pixel oder Register landet, gibt 0 zurück. Ein `poke` hat an dieser Stelle keine Auswirkung. Sprite-Flags, SFX- und Datentabellen befinden sich nicht in diesem Fenster. Verwenden Sie stattdessen `fget`, `sfxget` und `dget`.

Praktische Anwendung

Alle Funktionen sind am Ende dieser Seite aufgelistet, zusammen mit praktischen Anwendungsbeispielen für die PIX37-Funktionen. Die Links unter den Beispielen führen zu weiteren Informationen über die jeweiligen Funktionen.



Sanfte Bewegung

Eine 128×128 Pixel große Spielkassette, von oben betrachtet. Sprite 1 ist bereits gezeichnet. Pfeile beschleunigen in jede Richtung, Reibung bremst ab. Der Sprite wird umgebrochen: Die Hälfte, die unten herausragt, wird oben gezeichnet, und das Gleiche gilt für die anderen Ränder.

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



Prüfen, ob der nächste Zug blockiert ist

Eine 64×64-Spielkassette. Der Boden und die rechte Wand sind bereits auf der Karte markiert (Sprite 3), und Flagge 0 ist auf diesem Sprite gesetzt. Pfeile bewegen Sprite 1. Die Anzeige oben bedeutet „blockiert", wenn der nächste Schritt ein festes Feld überlappen würde, und „frei", wenn dies nicht der Fall wäre.

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



Ein Wisch mit Flip

Drücken Sie A. Der Wischvorgang wird durch flip() zeilenweise (schwarz) anstatt innerhalb von _draw() ausgeführt. _draw bleibt währenddessen im Hintergrund, sodass jede Zeile auf dem Bildschirm sichtbar bleibt.

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



Eine Tabelle mit Funktionen (Klasse)

Sprite 1 ist bereits gezeichnet. `make_monster` funktioniert wie eine Klasse. Es gibt eine Tabelle zurück, und die Methoden `update` und `draw` speichern die x- und y-Koordinaten des jeweiligen Monsters. Die Zahl oben gibt die zurückgelegte Strecke an.

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



Vier Möglichkeiten, einen Knopf zu lesen

A ist Taste 4. Halten Sie sie gedrückt, um den Ladestand zu erhöhen. Lassen Sie sie los, um den Ladestand auf 0 zurückzusetzen. Drücken Sie die Taste gedrückt, um die Menüleiste weiterzubewegen. „btnd" markiert den ersten Frame eines Tastendrucks, der von diesem Spielmodul ausschließlich als Beginn des Gedrückthaltens verwendet wird.

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



Mit der Maus schießen

Sprite 1 ist bereits in der Mitte gezeichnet. Ein weißer Punkt stellt den Cursor dar. Ein Linksklick verschießt einen Pixel in seine Richtung. Das Rad ändert die Geschwindigkeit, die in der Ecke angezeigt wird.

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



Geben Sie einen Namen ein und bestätigen Sie ihn.

Drücken Sie A, um das Feld zu öffnen. Mit den Pfeiltasten nach oben und unten ändern Sie einen blinkenden Buchstaben. A schreibt, X löscht, Y erzeugt ein Leerzeichen und B bestätigt. Anschließend wird Ihr Name angezeigt.

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



Kamera auf den Spieler

Eine 128×128 Pixel große Spielkassette. Sprite 1 befindet sich zu Beginn in der Mitte eines gefliesten Bodens. Pfeile bewegen sich. Die Kamera hält den Sprite zentriert. Die x- und y-Koordinaten in den Ecken scrollen nicht, da `camera()` ohne Argumente die Ansicht für das HUD zurücksetzt.

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



Ein paar Zeilen aus einem visuellen Roman

Die Dialoge sind in den Daten enthalten, auf Englisch und Polnisch. Mit A wird fortgefahren oder „Bleiben“ ausgewählt, wenn eine Auswahlmöglichkeit angezeigt wird. B wählt „Verlassen“ bei einer Auswahlmöglichkeit und wechselt ansonsten die Sprache. X wechselt die Sprache, während eine Auswahlmöglichkeit angezeigt wird. Der Dialog verwendet die Schriftart 8px, sodass polnische Buchstaben korrekt dargestellt werden.

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



Malen mit der Peek-and-Poke-Methode

Eine 64×64-Spielkassette, wobei Adresse 0 das Pixel oben links ist und die nächste Zeile bei 64 beginnt. Die Spielkassette schreibt Pixel, kopiert eine Zeile, löscht eine Zeile und gibt die gelesenen Werte aus.

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



Eine Liste, die Sie hinzufügen können

A addiert eine Zahl bis maximal 8. B entfernt die letzte. X entfernt die erste. Die Liste wird auf dem Bildschirm ausgegeben.

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



Ein Name auseinandergenommen

Die Spielkassette teilt ada,12,blue auf und druckt den Namen, seinen ersten Buchstaben, 12 und das Wort blue.

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



Spiele einen Ton, dann ändere ihn.

Muster 0 besteht aus einer einzigen Note und wird bei jeder Wiederholung gleich laut wiedergegeben. A spielt es erneut, sobald die Note verklungen ist. B erhöht die Tonhöhe und spielt diese Note.

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



Ein Lied, das in die Spielkassette geschrieben wurde.

Zwei Muster werden in Zeile 0 wiederholt, jeweils eines auf jedem der ersten beiden Kanäle. Die Zeile bleibt bei 0. ch0 und ch1 bezeichnen die Note, auf der sich der jeweilige Kanal befindet. Durch Bearbeiten der Parameter „Muster“, „Länge“ und „Schleife“ im Spielmodul lässt sich die Funktion dieser Regler im Musikfenster beobachten.

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



Drei Monster in DATA

Das Datenfeld enthält bereits eine Namensspalte und eine HP-Spalte mit den Einträgen Schleim, Fledermaus und Auge. A verletzt den Schleim. Die Zahlen rechts geben seine HP an.

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



Merkt euch das Ergebnis

A addiert einen Punkt und speichert ihn. Spiel beendet und öffnet die Spielkassette erneut: Die Punktzahl ist noch da. B vergisst sie.

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




Ein zweites Kapitel in einer anderen Datei

Zwei Spielmodule teilen sich den Speicherstand. Teil 1: A markiert das Spiel als abgeschlossen. Teil 2, der zuerst geöffnet wird, fordert dazu auf, Teil 1 zu beenden. Danach öffnet sich Teil 2, und B löscht den Speicherstand, sodass Teil 1 erneut gespielt werden kann.

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



Eine Errungenschaft

Sprite 8 ist bereits gezeichnet. Drücke zehnmal A. Die Benachrichtigung nennt den Erfolg, und Sprite 8 bleibt erhalten, da er freigeschaltet ist. Drücke B, um den Erfolg in der lokalen Datei zu löschen und den Zähler zurückzusetzen.

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



Ein Zimmer, das man betreten kann

Eine 256×256-Spielkassette. Ein weißer Punkt dient als Cursor. Das Menü bietet die Optionen „Host", „Beitreten" und „Serverliste". Der Host öffnet einen Raum namens „Walk" und startet das Spiel. „Beitreten" zeigt eine Adresse (Ziffern, Punkt, Doppelpunkt) an, und B verbindet sich. Die Serverliste wird mit den Pfeiltasten nach oben und unten bewegt, und A reiht sich in die Zeile ein. Im Spiel bewegt man mit den Pfeiltasten Sprite 1. Eine Münze wird geteilt: Der Host bewegt sie, und wer sie aufnimmt, erhält 1 Punkt mehr und der nächste Sprite erscheint. Jeder Sprite, auch der eigene, zeigt den aktuellen Punktestand an. In der Ecke werden die Anzahl der Spieler, die LAN-Adresse und die öffentliche Adresse des Hosts angezeigt. Verlässt B den Raum, kehren alle Spieler zum Menü zurück, und der nächste Host startet einen neuen Raum mit 0 Punkten.

`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 wartet

A zeigt die nächste Zeile an. Die Funktion pausiert an jeder Zeile, bis Sie A erneut drücken.

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



Warten Sie eine halbe Sekunde

Das Wort ändert sich nach etwa einer halben Sekunde von „…" zu „hi". Der Bildschirm zeichnet weiter, während die Coroutine pausiert.

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



Kreise, die nacheinander eintreffen

Fünf Kreise tropfen von oben herab, jeweils zwei Bilder voneinander entfernt, und fallen dann zu Boden.

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



Zwei Bilder, von der Uhr

Sprite 1 und Sprite 2 sind zwei Einzelbilder desselben Charakters. Das Bild wechselt viermal pro Sekunde.

Jede Funktion

Argumente in eckigen Klammern sind optional. Tabellen- und Zeichenkettenindizes beginnen bei 1, außer Datenzeilen und -spalten, die bei 0 beginnen. Farben sind Palettenindizes.

Index

_zeichnen
_init
_mp
_mp_fail
_mp_join
_mp_leave
_mp_update
_set_fps
_aktualisieren
Bauchmuskeln
achclear
achdef
erreichen
erreicht
hinzufügen
alle
behaupten
atan2
Band
bnot
bor
btn
Übrigens
btnp
btnu
bxor
Kamera
Decke
chr
Zirkel
Kreisfüllung
klickl
clickm
clickr
Clip
cls
mitgestalten
Farbe
coresume
cos
Costatus
zählen
Cursor
dcols
del
Feinkost
delsave
dget
Drows
dset
dstr
fget
füllen
umdrehen
flr
für jeden
fset
Infos
ipairs
Schlüssel aus
Keyon
keystat
Schlüsselstr
Linie
laden
Protokoll
lshr
Karte
max
memcpy
memset
mget
Mitte
min
Maus
mset
Musik
Musikstat
nannounce
ncall
als nächstes
nhost
nbeitreten
nkb
nkbstr
nlan
nverlassen
nlist
nhör zu
Hinweis
nplayers
nsenden
nstun
ord
oval
ovalfill

Packung	rotl	shl
Paare	rotr	Showstatistiken
Kumpel	Richtig	shr
palrgb	Richtig ausfüllen	Sünde
palt	sparen	Songget
pcall	Speicherslot	Liederset
Guck mal	Bildschirm	geteilt
peek2	scrollen	Spr
peek4	auswählen	Quadrat
perf	Setmap	Sand
pget	setpalette	sset
stupsen	Setspr	sspr
poke2	settex	stat
poke4	setturn	sub
drucken	sfx	Zeit
print10	sfxget	tline
print8	sfxhdr	tonum
pset	sfxset	tostr
rect	sfxstat	auspacken
rectfill	get	Whoami
Runde	sgn	Ertrag

Lebenszyklus

_init

`_init() ... end`

Wird einmalig beim Start der Spielkassette ausgeführt. Code, der sich außerhalb jeglicher Hooks befindet, wird ebenfalls in _init eingebunden.

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

_update

`_update() ... end`

Spiellogik. Wird mit 60 Hz aufgerufen, selbst wenn die Zeichenrate begrenzt ist oder der Host langsamer ist. Hier werden die Schaltflächen ausgelesen.

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

_draw

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

Zeichnet ein Bild. Ohne Argument oder mit dem Wert 0 wird die Funktion so oft ausgeführt, wie der Host Bilder sendet. Mit n (1..999) kann die Anzahl der Bilder pro Sekunde begrenzt werden. Die Aktualisierungsrate (_update) bleibt bei 60 Hz.

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

_set_fps

`_set_fps(n)`

Ändert die Zeichenbegrenzung (_draw cap) während des Spielablaufs. 0 bedeutet unbegrenzt. n ist auf 0 bis 999 begrenzt.

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

flip

```
flip()
```

Zeigt den Bildschirm an und wartet bis zum nächsten Frame. Verwenden Sie diese Funktion nur, wenn Sie eine eigene Frame-Schleife zeichnen. Führen Sie anschließend keine Busy-Wait-Anweisung aus.

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

time

```
time()
```

Sekunden seit _init.

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

Eingang

btn

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

1, solange Taste B gedrückt wird, sonst 0. p ist der Spieler (0–7). Ohne Argumente gibt die Funktion eine Bitmaske von Spieler 0 zurück. Tasten: 0 links, 1 rechts, 2 oben, 3 unten, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 Auswählen, 11 Start.

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

btnd

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

Trifft nur im _update-Frame zu, wenn die Taste b gedrückt wird.

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

btnp

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

Beim ersten Drücken wird die Taste aktiviert, dann erneut beim Gedrückthalten: 30 Ticks Verzögerung, danach alle 8 Ticks. Ohne Betätigung der Taste wird eine Maske angezeigt.

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

btnu

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

Trifft nur auf den Moment zu, in dem der Knopf losgelassen wird.

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

mouse

```
mouse()
```

Gibt die Mausposition x, y in Spielpixeln zurück.

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

clickl

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

Linke Maustaste. clickl() wird gedrückt gehalten. clickl(1) ist der Anschlag.

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

clickr

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

Rechte Maustaste. Gleiche 1-Argument-Kante wie clickl.

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

clickm

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

Mittlere Taste oder ein Klick auf das Scrollrad.

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

scroll

```
scroll()
```

Die Mausradbewegung in diesem Frame. Positiv und negativ hängen von der Drehrichtung des Rades ab.

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

keyon

```
keyon()
```

Öffnet ein Textfeld, das über Schaltflächen gesteuert wird. Mit den Pfeiltasten (nach oben/unten) kann ein Buchstabe (A–Z) ausgewählt werden. A schreibt den entsprechenden Buchstaben. X löscht. Y erzeugt ein Leerzeichen. B bestätigt und schließt das Feld. Die Tastatur gibt keine Daten in das Feld ein.

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

keyoff

`keyoff()`

Blendet das Textfeld aus.

```
keyoff()
```

keystr

`keystr([s])`

Text im Feld, dann 1 im Rahmen B zur Bestätigung, anschließend ändern sich die Buchstaben nach oben und unten. Dieser Buchstabe ist leer, sobald das Feld geschlossen wird. Mit `keystr("")` wird der Text gelöscht.

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

keystat

`keystat()`

1, solange das Textfeld geöffnet ist, ansonsten 0.

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

Zeichnung

screen

`screen()`

Aktuelle Bildschirmgröße der Spielkassette in Pixeln. Gibt Breite und Höhe zurück. Dies ist die Spielauflösung, nicht die des Editorfensters.

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

cls

`cls([col])`

Löscht den Bildschirm. Standardfarbe ist 0.

```
cls(1)
```

color

`color(col)`

Legt die Zeichenfarbe fest, die verwendet wird, wenn bei einem späteren Aufruf die Spalte weggelassen wird. Ohne Argumente wird die aktuelle Farbe zurückgegeben.

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

camera

`camera([x, y])`

Bei jedem Zeichenaufruf wird ein Offset vorgenommen. camera() wird auf 0, 0 zurückgesetzt.

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

clip

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

Beschränkt das Zeichnen auf ein Rechteck. Ein fünftes Argument ungleich null schneidet den aktuellen Clip. `clip()` stellt den Vollbildmodus wieder her. Die Ansicht der Clip-Register ist schreibgeschützt.

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

cursor

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

Legt fest, wo der nächste Ausdruck ohne Koordinaten erfolgen soll, und optional die Farbe. cursor() gibt x, y zurück.

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

pset

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

Setzt einen Bildschirmpixel. Wird in die Pixelstatistik einbezogen.

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

pget

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

Farbe des Bildschirmpixels an der Position x, y.

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

print

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

Zeichnet 3×5-Text (UTF-8, aus der Fusion-Schriftart ausgeschnitten). Ohne x- und y-Koordinaten wird an der Cursorposition gedruckt und in die nächste Zeile gewechselt.

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

log

`log(...)`

Fügt die durch Leerzeichen getrennten Argumente einem Textpuffer hinzu. pix37_run gibt diesen Puffer auf stdout aus. Es wird nichts gezeichnet.

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

print8

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

Gleiches gilt für den Druck, jedoch mit der 8 px Fusion-Schriftart.

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

print10

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

Gleiches gilt für den Druck, jedoch mit der 10 px Fusion-Schriftart.

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

line

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

Linie von x0,y0 nach x1,y1. Zwei Argumente setzen die Kette am vorherigen Ende fort. line() setzt diese Kette zurück.

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

circ

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

Kreisumriss. Radius in Pixeln.

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

circfill

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

Ausgefüllter Kreis.

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

oval

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

Ellipsenumriss innerhalb des Begrenzungsrahmens. Ecken sind eingeschlossen.

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

ovalfill

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

Ausgefüllte Ellipse in einem Begrenzungsrahmen.

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

rect

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

Rechteckige Kontur. Ecken sind eingeschlossen.

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

rectfill

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

Ausgefülltes Rechteck.

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

rrect

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

Abgerundeter Rechteckumriss. x, y ist die obere linke Ecke. w und h sind die Größe in Pixeln; beide müssen größer als 0 sein. r ist der Eckenradius und hat standardmäßig den Wert 0. Er ist auf die Hälfte der kürzeren Seite begrenzt.

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

rrectfill

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

Ausgefülltes abgerundetes Rechteck. Gleiche Argumente wie bei rrect.

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

fillp

`fillp([p])`

16-Bit-Füllmuster für später gefüllte Formen. 0 entspricht einer Vollfläche. `fillp()` gibt das aktuelle Muster zurück. Die Bits durchlaufen das 4×4-Muster.

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

pal

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

Ordnet die Farbe c0 der Farbe c1 zu. Modus 0 ist die Zeichenpalette, 1 die Anzeigepalette, 2 die Füllmusterpalette. pal() setzt die Einstellungen zurück.

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

palrgb

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

Setzt einen Paletteneintrag auf RGB, jeder Kanal 0..255.

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

palt

`palt([col, t])`

Transparenz. `palt(col)` oder `palt(col, true)` macht `col` transparent. `palt(col, false)` macht es deckend. `palt()` macht alle Farben undurchsichtig.

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

setpalette

`setpalette(n)`

Wählt die Palettenbank n aus. setpalette() gibt die aktuelle Bank zurück.

```
setpalette(1)
```

tline

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

Zeichnet eine Linie auf dem Bildschirm, während Kartenpixel von der Kartenposition mx, my, mdx, mdy abgetastet werden. Die Kartenabtastung erfolgt schrittweise entlang der Linie.

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

Sprites und Karten

spr

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

Zeichnet Sprite n an den Koordinaten x, y. w und h geben die Größe in Pixeln an (Standardwert: 8). fx und fy spiegeln horizontal bzw. vertikal, sofern sie ungleich null sind. tex ist der Sprite-Sheet-Tab. rot gibt den Drehwinkel in Umdrehungen (0 bis 1) an, analog zu sin und cos. Der Wert 0 bewirkt, dass der Sprite nicht gedreht wird.

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

sspr

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

Kopiert ein Rechteck des Sprite-Sheets auf den Bildschirm, skaliert auf dw mal dh. Standardmäßig bleibt die Quellgröße erhalten.

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

sget

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

Farbe eines Texels auf dem Sprite-Sheet.

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

sset

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

Schreibt ein Texel auf das Sprite-Sheet.

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

fget

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

Sprite-Flags. fget(n) gibt das Flag-Byte zurück. fget(n, f) ist wahr, wenn das Flag-Bit f (0..7) gesetzt ist.

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

fset

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

Setzt Sprite-Flags. fset(n, v) schreibt das gesamte Byte. fset(n, f, v) setzt ein Bit.

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

setspr

```
setspr(n)
```

Wählt das aktive Sprite-Sheet aus. setspr() gibt das aktuelle Sheet zurück. Gleicher Speicherbereich wie settex.

```
setspr(1)
```

settex

```
settex(n)
```

Alias von setspr.

```
settex(0)
```

map

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

Zeichnet die Karte. Zellen cx und cy der Karte werden an den Bildschirmpositionen sx und sy gezeichnet. cw und ch = 0 bedeuten die gesamte Karte. Die Ebene wird akzeptiert und ignoriert. tex ist das Tileset-Sheet.

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

mget

`mget(cx, cy)`

Sprite-ID in Map-Zelle cx, cy.

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

mset

`mset(cx, cy, n)`

Schreibt Sprite n in die Map-Zelle cx, cy.

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

setmap

`setmap(n)`

Wählt die Karte n aus. setmap() gibt die aktuelle Karte zurück. Karte 0 ist eine gültige Karte.

```
setmap(1)
```

Erinnerung

peek

`peek(addr)`

Liest ein Byte des 64 KB großen Fensters. Das Fenster besteht aus dem Live-Bildschirm, dann aus Sprite-Pixeln und schließlich aus Map-Zellen, wenn der Bildschirm in 65280 Bytes passt. Größere Bildschirme werden mit `poke(0x5F40, page)` seitenweise geladen. Nicht gemappte Adressen geben 0 zurück. Wird auch als `@addr` geschrieben.

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

poke

`poke(addr, ...)`

Schreibt Bytes an Adresse addr. Zusätzliche Argumente schreiben die folgenden Bytes. Nicht zugeordnete Adressen werden ignoriert. Außerdem werden $addr und val geschrieben. Das Antippen des Bildschirms zählt als Pixel-Schreibvorgang.

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

peek2

`peek2(addr)`

Vorzeichenloser 16-Bit-Wert, Little-Endian, an Adresse.

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

poke2

`poke2(addr, val)`

Schreibt einen vorzeichenlosen 16-Bit-Wert im Little-Endian-Format.

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

peek4

`peek4(addr)`

Vorzeichenloser 32-Bit-Wert, Little-Endian.

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

poke4

`poke4(addr, val)`

Schreibt einen vorzeichenlosen 32-Bit-Wert im Little-Endian-Format.

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

memcpy

`memcpy(dst, src, len)`

Kopiert len Bytes innerhalb desselben 64 KB-Fensters.

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

memset

`memset(dest, val, len)`

Füllt len Bytes ab dest mit val.

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

Mathe

abs

`abs(x)`

Absolutwert.

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

flr

`flr(x)`

Rundet sich gegen minus unendlich.

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

ceil

`ceil(x)`

Rundet gegen plus unendlich.

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

max

`max(a, b)`

Der größere der beiden Werte.

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

min

`min(a, b)`

Der kleinere der beiden Werte.

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

mid

`mid(a, b, c)`

Mittlerer der drei Werte. Als Klemme nützlich.

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

sgn

`sgn(x)`

Vorzeichen von x: -1, 0 oder 1.

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

sqrt

`sqrt(x)`

Quadratwurzel. Ein negatives Argument ist keine Zahl.

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

sin

`sin(x)`

Sinus. x ist abwechselnd: 0, 0,25, 0,5, 0,75, wobei ein Kreis einmal durchlaufen wird. Nicht im Bogenmaß.

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

cos

`cos(x)`

Kosinus wiederum, gleiche Konvention wie Sinus.

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

atan2

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

Winkel des Vektors dx, dy, abwechselnd von 0 bis 1.

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

rnd

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

Zufallszahl im Intervall [0, x). Standardmäßig ist x gleich 1, daher ist rnd() gleich 0..1. Für einen Würfel: flr(rnd(6))+1.

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

srand

```
srand(seed)
```

Initialisiert den Zufallsgenerator, sodass eine spätere rnd-Sequenz wiederholt werden kann.

```
srand(1)
```

band

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

Bitweises UND. Auch als a & b geschrieben.

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

bor

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

Bitweises ODER. Auch geschrieben als a | b.

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

bxor

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

Bitweises XOR. Auch geschrieben als a ^^ b.

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

bnot

```
bnot(a)
```

Bitweises NOT. Auch geschrieben ~a.

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

shl

```
shl(a, n)
```

Nach links verschieben. Auch als << n geschrieben.

shr

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

Arithmetische Rechtsverschiebung. Auch als >> n geschrieben.

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

lshr

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

Logische Rechtsverschiebung. Auch als >>> n geschrieben.

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

rotl

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

Drehe die Bits nach links. Wird auch als <<> n geschrieben.

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

rotr

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

Drehe die Bits nach rechts. Auch geschrieben als >>< n.

Tabellen und Werte

add

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

Fügt v an Tabelle t an. Falls i angegeben ist, wird es an diesem 1-basierten Index eingefügt.

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

del

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

Entfernt den ersten Eintrag, der gleich v ist.

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

deli

`deli(t, [i])`

Entfernt den Eintrag an Index i und gibt dessen Wert zurück. Ohne Angabe von i wird der letzte Eintrag entfernt.

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

all

`all(t)`

Iterator über den Array-Teil, der Reihe nach. Löcher beenden den Durchlauf analog zum Ende eines 1-basierten Arrays.

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

foreach

`foreach(t, fn)`

Ruft fn(v) für jedes Array-Element auf.

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

pairs

`pairs(t)`

Iterator über alle Schlüssel und Werte, einschließlich benannter Felder.

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

ipairs

`ipairs(t)`

Iterator über den Array-Teil mit 1-basiertem Index.

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

next

`next(t, [k])`

Gibt den nächsten Schlüssel und Wert nach k zurück. next(t) beginnt am Anfang. Wird von Paaren verwendet.

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

count

`count(t, [v])`

Länge des Array-Teils. Falls v angegeben ist, zählt die Funktion, wie viele Einträge gleich v sind.

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

pack

`pack(...)`

Die Argumente werden in einer neuen Tabelle zusammengefasst.

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

unpack

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

Gibt t[i] bis t[j] als separate Werte zurück. Standardmäßig wird der gesamte Array-Teil berücksichtigt.

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

select

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

`select("#", ...)` gibt die Anzahl der folgenden Argumente zurück. `select(i, ...)` gibt die Argumente ab i zurück.

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

assert

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

Die Ausführung wird mit einer Fehlermeldung abgebrochen, wenn c den Wert false oder nil hat. msg ersetzt den Standardtext.

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

pcall

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

Die Funktion f ruft bei Erfolg „true“ und das Ergebnis auf. Im Fehlerfall gibt sie „false“ und die Fehlermeldung zurück.

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

Saiten

chr

`chr(n)`

Ein Zeichen aus dem Bytecode 0..255.

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

ord

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

Bytecodes von s von Index i bis j. Die Indizes beginnen bei 1. Standardmäßig wird das erste Zeichen verwendet.

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

sub

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

Teilstring von i bis j, beginnend mit 1. Lass j weg, um den Rest zu erhalten.

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

split

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

Teilt s in eine Tabelle auf. sep ist standardmäßig ein Komma. Ein numerisches sep gibt die Blockbreite an. conv wandelt Teile mit tonum um, wenn es auf true gesetzt ist.

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

tonum

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

Parst eine Zahl. Die Basis kann zwischen 2 und 36 liegen. Gibt nil zurück, wenn der Text keine Zahl ist.

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

tostr

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

Wandelt einen Wert in Text um. Mit digits werden die Dezimalstellen für Zahlen festgelegt.

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

Audio

sfx

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

Spielt das Muster n ab und gibt den Kanal zurück. Wird ch weggelassen oder -1 übergeben, wird der erste freie Kanal gewählt. ch - 2 stoppt das Muster n auf allen Kanälen. n - 1 stoppt einen Kanal oder, wenn ch weggelassen wird, alle Kanäle. n - 2 lässt ein sich wiederholendes Muster zu Ende laufen, anstatt es erneut zu wiederholen. sfx() ohne Argumente gibt -1 zurück.

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

sfxget

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

Gibt eine Tabelle mit einem Pattern-Schritt zurück: [1] Tonhöhe, [2] Wellenform, [3] Lautstärke, [4] Effekt. Die Tonhöhe liegt zwischen 0 und 63 (C-0 bis D#5). Die Wellenform liegt zwischen 0 und 15. Die Lautstärke liegt zwischen 0 und 7, wobei 0 für Stille steht. Der Effekt liegt ebenfalls zwischen 0 und 7.

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

sfxset

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

Schreibt einen Musterschritt. Alle sechs Argumente sind erforderlich.

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

sfxhdr

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

Pattern-Header. `sfxhdr(n)` gibt die Header-Tabelle zurück. `sfxhdr(n, key, val)` setzt ein Feld: Geschwindigkeit, Loop-Einlass, Loop-Auslass, Pegel, Rauschen, Crush, Dtune, Hall, Dämpfung, Swing, Schritte. Zahlen von 0 bis 10 wählen dieselben Felder in dieser Reihenfolge aus. Die Geschwindigkeit wird in Ticks pro Schritt (TIC) angegeben. Höhere Werte bedeuten eine langsamere Geschwindigkeit. 15 entspricht 120 Schlägen pro Minute. 16-Bit-Synthesizer akzeptieren Werte von 1 bis 255. Schmalere Synthesizer passen sich automatisch ihrer Tick-Liste an.

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

sfxstat

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

Das auf Kanal ch abgespielte Muster ist die ID -1. Standardkanal ist 0.

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

note

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

Spielt einen einzelnen Ton ohne Muster. Tonhöhe 0–63, Wellenform 0–15, Lautstärke 0–7, Effekt 0–7. „Ticks“ gibt die Dauer an, im gleichen 120-Hz-Takt wie bei „TIC“. Ohne „Ticks“ dauert der Ton 15 Takte (ein Standardschritt). Der Effekt läuft während dieser Zeit, danach ist der Kanal stumm.

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

music

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

Startet Song n von oben. `music(-1)` stoppt die Wiedergabe. Wird `music` mit der ID eines bereits laufenden Songs aufgerufen, läuft dieser weiter. `music()` gibt die ID des aktuell gespielten Songs oder -1 zurück. `fade` wird in Millisekunden angegeben: Der Song wird ein- bzw. ausgeblendet, wenn `n` gleich -1 ist. Die gesetzten Maskenbits geben die für den Song verwendeten Kanäle an. 0 bedeutet alle acht Kanäle. Die übrigen Kanäle bleiben für Soundeffekte frei.

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

musicstat

```
musicstat()
```

Tabelle des aktuell abgespielten Liedes. „song“ und „row“ bezeichnen die Lied-ID und die Zeile (oder -1, wenn kein Lied abgespielt wird). „step“ und „pos“ bestehen aus jeweils acht Zahlen. „step[1]“ entspricht Kanal 0, „step[8]“ Kanal 7. „step“ ist der Notenindex (oder -1, wenn der entsprechende Kanal stumm ist). „pos“ gibt an, wie weit die Note von 0 bis 1 zurückgelegt wurde. „tick“ zeigt an, wie viele Ticks diese Zeile bereits abgespielt hat. „played“ gibt an, wie viele Zeilen seit dem Aufruf von „music()“ begonnen wurden, einschließlich der aktuell abgespielten.

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

songget

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

Die Muster-ID ist in einer Songzelle gespeichert. ch ist 0..7. 255 bedeutet, dass die Zelle leer ist. Dies ist der Live-Aufruf; der Code-Hinweis zeigt noch eine ältere Argumentreihenfolge an.

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

songset

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

Schreibt eine Muster-ID in eine Songzelle. Verwenden Sie 255, um die Zelle zu löschen.

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

stat

```
stat(n)
```

Nur Audio. 16–23 ist die SFX-ID auf Kanal n–16. 24–31 ist das Musikmuster auf Kanal n–24. Jeder andere Wert für n liefert 0. Frame-Timing und RAM-Nutzung sind Leistungs- und Informationsdaten, keine Statusdaten.

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

Datentabellen

dget

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

Liest eine Zelle der Datentabelle t aus dem Datenbereich. Zeile und Spalte beginnen bei 0.

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

dset

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

Schreibt eine Zelle. Verwenden Sie eine Zeichenkette für eine Textspalte.

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

dstr

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

Die Zelle wird als Text angezeigt. Zahlen werden umgewandelt.

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

drows

```
drows(t)
```

Wie viele Zeilen hat die Tabelle t?

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

dcols

```
dcols(t)
```

Wie viele Spalten hat die Tabelle t?

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

Speichern und Erfolge

save

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

Speichert v unter dem Schlüssel k in der Speicherdatei. v kann eine Zahl, eine Zeichenkette oder eine Tabelle davon sein. Die Datei wird komprimiert.

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

load

`load(k)`

Liest den Schlüssel k. Gibt nil zurück, wenn der Schlüssel fehlt.

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

delsave

`delsave(k)`

Löscht den Schlüssel k aus dem Speicherstand.

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

saveslot

`saveslot([name])`

Wählt die Speicherdatei im Ordner des Autors aus. Ohne Namen wird der Spieltitel als Speicherort verwendet. Gibt 1 zurück, wenn die Datei bereits existiert.

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

achieve

`achieve(i)`

Schaltet die Errungenschaft i frei (0 bis 15). Lokal gespeichert. Ein Steam-Build setzt ebenfalls ACH_i. Eine Katalogsitzung meldet dies dem Spielerkonto.

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

achieved

```
achieved(i)
```

1. Wenn Erfolg i bereits freigeschaltet ist.

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

achdef

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

Toast-Text für Erfolg i, der 15 Sekunden lang vom unteren Bildschirmrand angezeigt wird. spr ist eine optionale 8×8-Kachel, tex ihr Blatt.

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

achclear

```
achclear(i)
```

Löscht den Erfolg i in der lokalen Datei. Ein bereits festgelegter Steam-Erfolg ACH_i bleibt bestehen.

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

Netzwerk

whoami

```
whoami()
```

Deine Spieler-ID (0–15) gilt während einer Sitzung. Offline ist sie -1.

```
id = whoami()
```

nhost

```
nhost()
```

1, wenn diese Maschine der Host ist, andernfalls 0.

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

nplayers

```
nplayers()
```

Wie viele Live-IDs gibt es im Bereich 0..15?

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

nlisten

```
nlisten()
```

Startet den UDP-Host und gibt die LAN-Adresse als IP:Port zurück.

```
addr = nlisten()
```

njoin

```
njoin(addr)
```

Verbindet sich mit IP oder IP:Port.

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

nleave

```
nleave()
```

Verlässt die Sitzung. Peers sehen _mp_leave mit dem Grund 0.

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

nsend

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

Sendet eine verlustbehaftete Nachricht. Der Name ist das Ereignis. Es folgen bis zu vier zusätzliche Werte. p von 0 bis 15 sendet die Nachricht an den betreffenden Peer; lässt man p weg, sendet man die Nachricht an alle.

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

ncall

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

Gleiche Struktur wie nsend, aber die Nachricht wird so lange wiederholt, bis sie bestätigt wird. Drei aufeinanderfolgende Fehlschläge beenden die Sitzung.

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

nlan

```
nlan()
```

Aktuelle LAN-IP:Port nach nlisten.

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

nstun

```
nstun()
```

Ihre öffentliche IP-Adresse und der Port, ermittelt von STUN. Ein Gast in einem anderen Netzwerk kann versuchen, sich damit zu verbinden.

```
pub = nstun()
```

setturn

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

TURN-Fallback, falls ein direkter Pfad fehlschlägt. setturn() mit leeren Argumenten deaktiviert TURN. Die Adresse hat das Format Host:Port.

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

nlist

```
nlist(game)
```

Tabelle der offenen Spielsitzungen für diesen Spielnamen. Jeder Eintrag enthält Adresse, Spieler und Name.

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

nannounce

```
nannounce(game)
```

Heartbeat, damit dieser Host in der Spielliste erscheint. Rufe ihn während des Host-Vorgangs per Timer auf.

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

nkb

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

Adressfeld. nkb(1) öffnet es, nkb(0) blendet es aus. nkb() zeigt an, ob es geöffnet ist. Pfeiltasten nach oben und unten ändern ein Zeichen (0–9, Punkt oder Doppelpunkt). A schreibt. X löscht. Die Tastatur gibt keine Daten in dieses Feld ein.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

Im Adressfeld wird Text angezeigt, wobei die Zeichen nach oben und unten geändert werden. nkbstr("") löscht den Text.

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

_mp

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

Sie schreiben diese Funktion. Sie wird ausgeführt, sobald eine Nachricht eintrifft. „from“ ist die Absender-ID. „name“ ist das Ereignis. Es können bis zu vier Werte folgen.

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

_mp_join

`_mp_join(p)`

Sie schreiben diese Funktion. Ein Peer hat die ID p erhalten. Sie wird auch für Sie ausgelöst; whoami() ist diese ID.

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

_mp_leave

`_mp_leave(p, why)`

Sie schreiben diese Funktion. Peer p hat die Verbindung verlassen. Warum steht 0 für ein normales Verlassen? Warum steht 1 für einen Verbindungsverlust?

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

_mp_update

`_mp_update()`

Sie schreiben diese Funktion. Sie wird alle 15 _update Frames ausgeführt, also etwa 4 Hz. Ein guter Ort, um nsend-Positionen zu übermitteln.

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

_mp_fail

`_mp_fail()`

Sie schreiben diese Funktion. Die Sitzung ist beendet: Der Host ist nicht mehr erreichbar oder ncall ist dreimal fehlgeschlagen.

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

Koroutinen

cocreate

```
cocreate(f)
```

Erzeugt eine Koroutine aus der Funktion f. Sie wird erst nach coresume ausgeführt.

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

coresume

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

Führt die Koroutine c aus, bis sie einen Wert liefert oder zurückkehrt. Im Erfolgsfall: true und die gelieferten Werte. Im Fehlerfall: false und die Fehlermeldung.

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

costatus

```
costatus(c)
```

"ausgesetzt", "laufend" oder "tot".

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

yield

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

Gibt die Kontrolle an denjenigen zurück, der coresume aufgerufen hat. Die Werte werden zu den Ergebnissen dieses coresume-Aufrufs.

```
yield(1)
```

Messen

info

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

Zeichnet das Live-HUD in _draw: fps (1-Sekunden-Durchschnitt), upd, drw, px, max, ram, peak, ramc und net, wenn eine Sitzung aktiv ist. info() verwendet den Druckcursor. info(col) legt nur die Farbe fest. info(x, y) oder info(x, y, col) platziert die Farbe. Zahlen verwenden B, k oder m, um auf kleinen Bildschirmen dargestellt zu werden. Der Aufruf selbst wird nicht in px gezählt.

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

showstats

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

Gleiches HUD wie die Infoanzeige, automatisch bei 0, 0 nach _draw gezeichnet, Kamera und Clip werden ignoriert. showstats(0) blendet es aus. showstats(1) oder showstats() zeigt es an.

```
showstats(1)
```

perf

```
perf(n)
```

Eine HUD-Nummer, damit Sie sie selbst zeichnen können. 0 fps, 1 upd, 2 drw, 3 px, 4 max (höchste Frame-Auslastung in dieser Sekunde), 5 RAM, 6 RAMc (RAM plus die Spielkassette), 7 maximaler RAM-Verbrauch seit Spielstart.

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