Instrukcja obsługi PIX37

Edytor, potem kod. Kopia offline książki na pix37.party/docs.

Edytor

PIX37 DEV to edytor. Rysujesz, piszesz i słuchasz gry, a następnie eksportujesz kasetę lub samodzielną kompilację. Program może również odtwarzać kasety i otwierać PIX37.PARTY, aby pobierać kasety i korzystać z katalogu. PIX37 DEV EDUCATIONAL to ten sam edytor, i jest darmowy. Nie oferuje kompilacji, eksportu kaset ani dostępu do PIX37.PARTY. PIX37 Console to odtwarzacz. Odtwarza kasety `.p37`, a Ty otwierasz PIX37.PARTY, aby pobierać kasety i przeglądać katalog.

Projekty i kasety

Plik `.pix37` to projekt: zakładki źródłowe, arkusze sprite'ów, mapy, dźwięk i tabele danych. Plik `.p37` to kaseta, którą odtwarza odtwarzacz: ściśle upakowany bajtkod.

Zapisy i osiągnięcia znajdują się w folderze autora obok danych aplikacji. W systemie Windows wygląda to tak: `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` dla zapisu i `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` dla osiągnięć. W systemie Linux te same pliki to `~/.local/share/PIX37/Mira/Pong.p37s` i `~/.local/share/PIX37/Mira/Pong.ach`. Ada to konto na komputerze. Mira to nazwa autora, a ta nazwa to folder. Pong to slot zapisu, domyślnie odpowiadający tytułowi gry. `save` zapisuje `.p37s`. Pierwsze odblokowane osiągnięcie zapisuje `.ach`. Jeśli pliki już tam są, gracz odczytuje je po uruchomieniu kasety.

Menu deweloperskie

Klawisz Escape otwiera menu deweloperskie. Klawisz Escape podczas gry wraca do edytora. Potwierdź Enterem lub przyciskiem A (domyślnie V). Wstecz to przycisk B (domyślnie C).

Jeśli nie załadowano żadnego projektu, menu wygląda następująco: UTWÓRZ PROJEKT, WCZYTAJ PROJEKT, BUDUJ, WCZYTAJ kasetę, POŁĄCZ Z PIX37.PARTY, USTAWIENIA.

Po załadowaniu projektu wykonaj następujące czynności: ZAPISZ PROJEKT, WCZYTAJ PROJEKT, UTWÓRZ PROJEKT, ZASOBY, ZBUDUJ, WCZYTAJ kasetę, POŁĄCZ Z PIX37.PARTY, USTAWIENIA.

Ctrl+S zapisuje otwarty projekt. Ctrl+R go uruchamia.

Utwórz projekt

CREATE PROJECT prosi o kasetę przed rozpoczęciem tworzenia projektu.

- **WIDEO** — 1-BIT (2 kolory), 2-BIT (4), 4-BIT (16), 8-BIT (256).
- **AUDIO** — głębokość próbki syntezatora: 16, 8, 4, 2 lub 1 bit.
- **SCREEN** — ustawienia wstępne obejmują 64×64, 128×128, 160×90, 256×256 i 320×180. Rozmiar CUSTOM pozwala wybrać dowolny rozmiar od 1×1 do 512×512.
- **PALETY** — banki kolorów. Pierwszy bank zaczyna się od poniższego 16-kolorowego zestawu. Funkcja WCZYTAJ PALETĘ stosuje wbudowany zestaw: PICO-8 16, PICO-8 32, Game Boy, 1-bit, CGA, Sweetie-16 oraz dłuższą listę zaczerpniętą z Lospec (Resurrect 64, Apollo, Endesga 32 i 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century i inne). Możesz również wczytać plik palety lub wczytać własny plik PNG i pobrać paletę z tego obrazu.



Indeksy kolorów używane przez `cls`, `pset`, `spr` i pozostałe. Możesz przemalować dowolny indeks w edytorze palet.

Pięć paneli

Edytor to GAME, PAINT, CODE, SOUND i DATA, w tej kolejności. Mapa znajduje się w PAINT. SFX i utwory znajdują się w SOUND. Kliknij panel, aby tam pracować. Gdy CODE jest aktywne, SOUND i DATA ignorują klawisze liter, więc pisanie nie powoduje jednoczesnego grania na pianinie.



> Pełny układ, zapisany `1:1:4:1:1`. Każda liczba określa, ile miejsca w wierszu zajmuje dany panel: GRA, FARBA, KOD, DŹWIĘK, DANE. Mały ekran znajduje się na środku komórki o wymiarach 128×128 pikseli. Ekran większy niż 128 pikseli poszerza kolumnę GRA.

USTAWIENIA → GLOBALNE → UKŁAD cyklicznie przełącza ustawienia:

Kod	Co widzisz
`1:1:4:1:1`	Wszystko. GRA i DŹWIĘK ułożone w stos, FARBA i DANE ułożone w stos, KOD po prawej.
`1:1:4:1:1 2X`	Ten sam układ. KOD jest rysowany z dwukrotnie większą gęstością, więc mieści się w nim dwa razy więcej tekstu.
`1:0:1:0:1`	GRA, DANE, KOD.
`1:0:1:0:1 2X`	Ta sama aranżacja, z gęstszym KODEM.
`0:1:1:1:1`	Brak podglądu gry. FARBA, KOD, DŹWIĘK, DANE.
`0:1:1:1:1 2X`	Ta sama aranżacja, z gęstszym KODEM.
`0:1:4:1:2`	Studio. Szeroki KOD, wyższy pasek DATA, FARBA i DŹWIĘK.
`0:1:4:1:2 2X`	Ta sama aranżacja, z gęstszym KODEM.
`0:0:4:0:4`	Tylko KOD i DANE, oba duże.
`0:0:4:0:4 2X`	Ta sama aranżacja, z gęstszym KODEM.
`0:1:4:1:0`	FARBA, DŹWIĘK i szeroki KOD.
`0:1:4:1:0 2X`	Ta sama aranżacja, z gęstszym KODEM.
`0:4:0:0:0`	Tylko PAINT, 256×256, jeden piksel arkusza na jeden piksel płótna.

Wiersz `2X` zachowuje te same panele. Panel CODE jest rysowany z dwukrotnie większą rozdzielczością i skalowany do tej samej komórki, dzięki czemu tekst, zakładki i margines są o połowę niższe, a na ekranie mieści się dwa razy więcej tekstu źródłowego.

GRA

Podgląd działającej kasety. Kliknij panel GRA lub naciśnij Ctrl+R, aby uruchomić podgląd. Panel konfiguruje się w USTAWIENIA → PANEL GRY:

- **Podgląd** — kolor bezczynności, ciągły lub automatyczne ponowne ładowanie podczas edycji.
- **Graj w nowym oknie** — gra otworzy się obok edytora.
- **Wyświetl** — podgląd jako RGB888 lub RGB565.
- **GRA 64** — ekran 64×64 jest albo wyśrodkowany 1:1 w komórce, albo skalowany 2×, aby ją wypełnić.
- **Shadery** — CRT i filtry w grze, niezależnie od filtrów w edytorze.
- **Kolor** — tło panelu.

FARBA

Arkusze sprite'ów oraz, po stronie mapy tego samego panelu, mapy kafelków. Arkusze mogą być większe niż ekran. Rozmiary podglądu sprite'ów to 8, 16, 32, 64 i 128 pikseli (128 tylko wtedy, gdy arkusz ma co najmniej 128 pikseli). Przesuń kółko myszy nad podgląd sprite'a, aby go powiększyć. Rozmiary pędzla to 1, 2, 3 i 4.

Klawisz	Narzędzie
B	Ołówek
M	Wybierz prostokąt
Q	Lasso
S	Stempel
G	Wiaderko
I	Kroplomierz
V	Przenosić
L / Shift+L	Linia / krzywa
U / Shift+U	Pusty prostokąt / pusta elipsa
F / Shift+F	Wypełniony prostokąt / wypełniona elipsa
D	Dygotać
[ ]	Mniejszy / większy pędzel
Ctrl+Z / Ctrl+Y	Cofnij/ponów. Ctrl+Shift+Z również powoduje ponowienie.
Ctrl+C / Ctrl+X / Ctrl+V	Kopiuj, wytnij, wklej
Ctrl+A	Wybierz płótno

Mapa wykorzystuje tę samą ideę, z mniejszą liczbą narzędzi: B ołówek, M zaznaczenie prostokąta, S stempel, V przesunięcie oraz Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Przytrzymaj klawisz Alt, aby powiększyć teksturę mapy. Rozmiar pędzla na mapie pozostaje taki sam, jak w przypadku narzędzi malarskich.

Każdy sprite ma osiem bitów flag. Paint ustawia je na sprite'ie; `fget` i `fset` odczytują je w grze.

USTAWIENIA → EDYTOR FARBY to miejsce, w którym konfiguruje się ten panel:

- **Paleta** — który bank jest narysowany, plus wczytanie i zapisanie.
- **Podgląd** — RGB888 lub RGB565.
- **Kursor pędzla** — wskaźnik używany podczas malowania.
- **Przyciąganie pikseli** — sposób, w jaki pędzel przyciąga się do pikseli.
- **Duży pędzel** — duży pędzel wyskakuje z lewego górnego rogu lub z całego swojego śladu.
- **Skórka cebuli** — duch innych ramek i kierunek, w którym zwrócony jest ten duch.
- **Powiększenie mapy** — prawdziwe piksele, czyli swobodne powiększenie.
- **Kolory** — kolory interfejsu użytkownika programu Paint.

KOD

Zakładki to oddzielne pliki w projekcie. Kliknij nazwę zakładki, aby ją zmienić; Enter potwierdza. Kilka zakładek, z których każda definiuje kod `_init`, `_update` lub `_draw`, jest łączonych w kolejności zakładek: pierwsza zakładka jest uruchamiana jako pierwsza, ostatnia jako ostatnia. Kod, który nie znajduje się w haku, staje się częścią `_init`. Kaseta bez żadnych haków jest traktowany jako jeden `_draw`.

Klawisz	Działanie
Strzałki	Przesuń kursor. Klawisz Shift rozszerza zaznaczenie.
Ctrl+Lewo / Prawo	Słowem.
Home / Koniec, Ctrl + Home / Koniec	Początek lub koniec linii lub tabulatora.
Patka	Wstaw dwie spacje. Na liście uzupełnień zaakceptuj nazwę.
Wchodzić	Nowy wiersz, wcięty jak poprzedni. Akceptuje uzupełnienie.
Ctrl+C / X / V	Kopiuj, wytnij, wklej.
Ctrl+Z / Ctrl+Y	Cofnij / ponów.
Ctrl+A	Wybierz kartę.
Ctrl+D	Duplikuje linię lub zaznaczenie w dół.
Ctrl+/	Skomentuj lub usuń komentarz za pomocą `--`.

Ctrl+F	Znajdź. F3 przeskakuje do następnego trafienia. Escape zamyka pasek.
Ctrl+G	Przejdź do numeru wiersza.
Ucieczka	Zamyka najpierw opcje „Znajdź”, „Przejdź do” lub „Lista podpowiedzi”. Drugie naciśnięcie klawisza Escape powoduje przejście z edytora do menu programistycznego.

Podpowiedzi pojawiają się podczas wpisywania nazwy funkcji. Enter lub Tab wstawia nazwę, a `(`, jeśli opcja COMPLETE (jest włączona w ustawieniach. USTAWIENIA → EDYTOR KODU:

- **Szukaj** — Ctrl+F na aktywnej karcie lub na każdej karcie.
- **Tekst wysoki** — przytnij glify do 5 pikseli lub pokaż cały glif.
- **Rozmiar ciągu** — rozmiar czcionki literałów ciągu.
- **Czcionka komentarza** — 3×5 lub czcionka Fusion bieżącego języka interfejsu użytkownika.
- **Complete (** — czy uzupełnianie wpisuje nawiasy.
- **Podpowiedzi do kodu** — włączone lub wyłączone.
- **Język wskazówek** — język menu lub angielski.
- **Caret space** — dodatkowe przesunięcie w pustych wierszach.
- **Margines** — numery wierszy.
- **Kolor** — tło, komentarze, tabulatory, liczby, operatory, słowa kluczowe, ciągi znaków, funkcje silnika, Twoje funkcje, zmienne, nawiasy, kursor, zaznaczenie, wiersz błędu, ostrzeżenie.

DŹWIĘK

Panel ma dwie zakładki. SFX to jeden wzór: krótki ciąg nut. SOUND to piosenka: umieszcza te wzory na ośmiu kanałach. TIC to liczba taktów, na które czeka jeden krok. Zegar ma 120 taktów na sekundę, a jeden krok to szesnastka, więc 15 taktów to 120 uderzeń na minutę. Wyższy TIC to wolniejszy wzór. Dwa wzory pozostają w tym samym kroku, gdy jeden TIC dzieli drugi: 15 i 30 tak, 15 i 8 nie.

Fortepian gra po wybraniu nuty myszką. Białe klawisze to ZXCVBNM (CDEFGAB). Czarne klawisze to SDGHJ (C#, D#, F#, G#, A#). Najedź kursorem na dowolną wartość i wpisz liczbę lub najedź na nią kursorem i przekręć kółko. Numery wysokości dźwięku w kodzie mieszczą się w zakresie od 0 (C-0) do 63 (D#5).

W jednym ze wzorów LOOP i LEN to te same dwie liczby. OUT powyżej 0 tworzy pętlę od IN do OUT, a tytuł brzmi LOOP. OUT przy 0 i IN powyżej 0 to LEN: tyle właśnie nut jest granych od początku, po czym wzór się kończy, a tytuł brzmi LEN. Oba przy 0 odtwarzają każdy krok.

Zakres edycji syntezatora zależy od wybranej głębokości bitowej kasety. USTAWIENIA → EDYTOR SFX ustawia tę głębokość, to, czy nuta ma automatycznie przeskakiwać (wyłączone, w łańcuchu nut lub w łańcuchu pól), czy klawisz Space wznawia odtwarzanie lub przełącza między startem i stopem, a także kolory edytora.

16-bit

Każdy krok ma przypisaną nutę, głośność, falę i efekt. Głośność wynosi od 0 do 7. 0 oznacza ciszę.

Fale:

- **0 trójkąt**
- **1 piła pochylona**
- **2 piły**
- **3 kwadraty**
- **4 impuls** — wąski impuls
- **5 organów**
- **6 hałas**
- **7 fazerów**
- **8 PWM** — kwadrat, którego szerokość się zmienia
- **9 puls 25%**
- **10 sinusów**
- **11 metaliczny hałas**
- **Dzwonek 12 FM**
- **13 zrywać**
- **14 supersaw**
- **15 PWM (C64)**

Efekty na pojedynczej nucie:

- **0 brak**
- **1 slajd** — wysokość i głośność przesuwają się do następnej nuty w tym kroku
- **2 vibrato** — wahania wysokości dźwięku
- **3 drop** — dźwięk spada do ciszy, zwykły kopniak
- **4 zanikanie**
- **5 wyciszenie**
- **6 arpeggio** — gra tę nutę i trzy kolejne, po jednej co 8 taktów. Przy TIC 8 lub niższym, co 4 takty.
- **7 szybkich arpeggio** — te same cztery nuty, co 4 ticki lub co 2, gdy TIC wynosi 8 lub mniej.

Nagłówek wzoru, nad notatkami:

- **TIC** — liczba uderzeń na krok, od 1 do 255. 15 to 120 uderzeń na minutę. Wpisz liczbę lub zmień ją o jeden.
- **LOOP IN / OUT** — z OUT powyżej 0, wzór zapętla się od IN do OUT. Pozostaw OUT na 0, a tytuł zmieni się na LEN: IN oznacza liczbę nut, a 0 oznacza cały wzór.
- **SZUM** — cały wzór staje się szumem.
- **CRUSH** — żywiołowa wersja każdej fali.
- **DTUNE** — 0 wyłączony, 1 lekki refren, 2 mocny refren.
- **RVERB** — pogłos: 0 wyłączony, 1 krótki, 2 długi.
- **DAMPN** — filtr dolnoprzepustowy: 0 wyłączony, 1 słaby, 2 mocny.

- **POZIOM** — wolumen całego wzoru, od 0 do 7.
- **SWING** — tasowanie. Nieparzyste kroki czekają trochę dłużej, parzyste trochę krócej.
- **KROKI** — liczba kroków użytych we wzorze: 32, 24, 16 lub 12.

8-bit

Krok składa się z nuty i fali. Cztery fale to kwadrat, trójkąt, piła i szum. Nagłówek zawiera TIC, LOOP IN/OUT, FX i LEVEL. TIC to jeden z następujących efektów: 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60. FX to jeden efekt dla całego wzoru: 0 – brak, 1 – wibrato, 2 – arpeggio, 3 – wyciszenie. LEVEL wynosi od 0 do 3.

4-bit

Krok to tylko nuta. Nagłówek zawiera TIC, LOOP (włączone lub wyłączone), WAVE (te same cztery fale) i OCT (oktawę, w której znajdują się nuty). TIC to jedna z oktaw: 4, 6, 8, 10, 12, 15, 20, 30.

2-bit

Krok jest pusty, bęben, uderzenie lub miażdżenie. Nagłówek ma TIC i BASE. TIC wynosi 8, 15, 16 lub 30. BASE określa, jak wysoko osiada bęben i uderzenie.

1-bit

Krok to nuta. TIC wynosi 8, 15, 16 lub 30, czyli te same cztery prędkości co w przypadku 2-bitów.

Piosenki

Karta DŹWIĘK to otwarty utwór. Osiem kolumn to kanały. Komórka to identyfikator wzorca. Pusta wartość 255 oznacza, że kanał w danym wierszu jest wyciszony. Kolumna X to polecenie: kontynuuj, zatrzymaj, rozpocznij pętlę lub zakończ pętlę. Zatrzymaj kończy utwór po wierszu. Koniec pętli powoduje powrót do ostatniego początku pętli.

Rząd kończy się, gdy skrajny lewy wzór z LOOP OUT na 0 się kończy. Wzory bardziej na prawo nie wydłużają rzędu: dłuższy jest odcinany, krótszy zaczyna się od początku, aż do końca rzędu. Wzór z LOOP OUT powyżej 0 zapętla się między LOOP IN a LOOP OUT i nie kończy rzędu. Jeśli wszystkie wzory w rzędzie zapętlają się, rząd pozostaje w tym miejscu.

Skróty

Klawisz	Działanie
Klawisze fortepianu	Napisz notatkę na temat kroku wybranego za pomocą myszki.
Góra / Dół	Zmień wartość pod kursorem.

Przestrzeń	Odtwórz otwarty efekt dźwiękowy (SFX) lub utwór z pierwszego rzędu. Uruchom/zatrzymaj lub zrestartuj w ustawieniach edytora efektów dźwiękowych (SFX EDITOR).
Ctrl+Z / Ctrl+Y	Cofnij / ponów.
Ctrl+C / Ctrl+X / Ctrl+V	Skopiuj, wytnij, wklej zaznaczenie.

DANE

Tabele nazwane. Każda tabela składa się z wierszy i kolumn. Z kodu `dget`, `dset`, `dstr`, `drows` i `dcols` korzystają z tych tabel. Numery wierszy i kolumn w kodzie zaczynają się od 0.

Typy kolumn cyklicznie wyświetlane w nagłówku:

- **bool** — jeden bit.
- **2** — dwa bity.
- **4** — cztery bity.
- **8** — bajt.
- **16** — 16 bitów, dla identyfikatorów i większych liczb.
- **spr** — identyfikator sprite'a.
- **str** — tekst narysowany czcionką systemową.
- **str8** i **str10** — ten sam tekst, narysowany wyżej.
- **f16** i **f32** — float półprecyzyjny i float pełny.

Gdy wskaźnik znajduje się na nagłówku kolumny, kombinacja Ctrl+przewijanie zmienia jej typ. Zwykłe przewijanie przesuwa się w poprzek kolumn lub w dół wierszy, w zależności od tego, co zostanie wybrane w USTAWIENIACH → EDYTOR DANYCH. Ctrl zmienia kierunek przewijania, dlatego kombinacja Ctrl+przewijanie w nagłówku zmienia typ, gdy kółko jest ustawione na kolumny.

Klawisze strzałek służą do poruszania się. Enter edytuje komórkę. USTAWIENIA → EDYTOR DANYCH:

- **Koło** — przewijanie kolumn lub wierszy.
- **Kolory** — siatka, nagłówek i zaznaczenie.

Aktywa

ASSETS importuje i eksportuje otwarty fragment, a nie cały projekt. Sprite'y używają sprite'a otwartego w programie PAINT na otwartym arkuszu. Mapa to otwarta mapa. SFX to otwarty wzór. Muzyka to otwarta piosenka. Dane to otwarta tabela. Kod jest wyjątkiem: każda karta jest zapisywana i każda karta jest odczytywana.

Uprzejmy	Plik	Co się rusza
Duszki	`.png`	Otwarty sprite.
Mapa	`.p37map`	Otwarta mapa.
Efekty specjalne	`.p37sfx`	Otwarty wzór.
Muzyka	`.p37mus`	Otwarta piosenka. Numery wzorów pozostają bez zmian. Wyeksportuj efekty dźwiękowe osobno, jeśli ich potrzebujesz.
Dane	`.p37dat`	Otwarty stół.
Kod	`.lua`	Każda zakładka.

Ustawienia

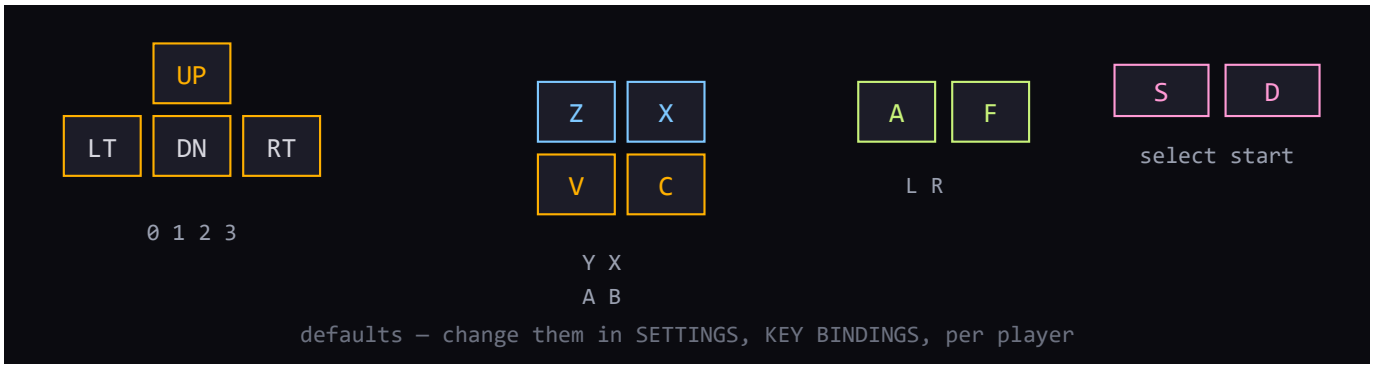
Lista główna:

Strona	Co przechowuje
PROFILE	Zapisz i wczytaj każde ustawienie. Nowe, zmień nazwę, nadpisz, usuń. Skopiuj lub wklej profil jako tekst.
Menu	Okno, klawisze, kolory menu, opcjonalne tło.
PANEL GRY	Okno podglądu i odtwarzania.
EDYTOR KODU	Podświetlenia i wskazówki.
edytor FARBY	Pędzel, paleta, widok mapy.
MONTAŻYSTA EFEKTÓW SKUTECZNYCH	Syntezator i wprowadzanie nut.
EDITOR DANYCH	Kierunek i kolory kół.
JĘZYK	Język menu i podpowiedzi.
LICENCJE	Teksty licencji obejmujące czcionkę Fusion Pixel i SDL3.

ŚWIATOWY:

- **UKŁAD** — układ paneli, powyżej.
- **GŁOŚNOŚĆ** — poziom główny, od 0 do 10.
- **PRZYPISY KLAWISZY** — klawisze dla każdego z 8 graczy. Naciśnij klawisz, aby go przypisać. Kierunki, ABXYLR, wybierz.

- **Prezentuj** — pomijaj nieaktywne klatki lub rysuj każdą klatkę.
- **Pełny ekran** — bez obramowania.
- **Odśwież** — liczba klatek na sekundę (FPS) w oknie. Krok w lewo i w prawo: 60, 90, 120 i maks. Enter wpisuje liczbę do 1000. 0 oznacza brak limitu.
- **Vsync** — czeka na wyświetlenie obrazu pomiędzy klatkami.
- **Skala** — mnożnik okna całkowitego.
- **Shadery** — CRT i filtry w edytorze, nie w grze.
- **Kolor** — tekst menu i podświetlenie.
- **Tło** — opcjonalny plik PNG, przycięty do rozmiaru 128, za menu. Wyczyść, aby je usunąć.
- **Kursor edytora** — kształt wskaźnika.
- **Mysz** — wskaźnik ruchomy lub przyciągany do pikseli.
- **Szybkość interfejsu użytkownika** — szybkość, z jaką wybrany kierunek jest powtarzany na liście.
- **Eksplorator** — listy plików zaczynają się w katalogu głównym komputera lub w folderze exe.



Domyślna klawiatura. A to potwierdzenie w menu. B to powrót. W grze są to przyciski 4 i 5. Select (S) to przycisk 10, a nie skrót menu.

Przycisk	Rola	Klucz domyślny
0	Lewy	Strzałka w lewo
1	Prawidłowy	Strzałka w prawo
2	W górę	Strzałka w górę
3	W dół	Strzałka w dół
4	A, potwierdź	V
5	B, z tyłu	C
6	X	X
7	Y	Z
8	R	F

9	L	A
10	Wybierać	S
11	Start	D

Kompilacja

Wybierz platformę: Windows, Linux, Mac, Android, iOS lub Web.

- **Nazwa** — nazwa pliku kompilacji.
- **Autor** — w systemach Windows i Linux. To nazwa folderu do zapisu.
- **Identyfikator aplikacji Steam** — w systemach Windows i Linux. Pusty oznacza brak Steamworks. Wypełniony identyfikator mapuje `achieve(0)`…`achieve(15)` na osiągnięcia Steam ACH_0…ACH_15.
- **Id** — identyfikator pakietu, o który platforma prosi.
- **Ukryj kod** — w nakładkach na komputery i urządzenia mobilne. Spowalnia kod `peek` w kasecie. Nie jest to blokada.
- **Pełny ekran** — wyeksportowany odtwarzacz uruchamia się w trybie pełnoekranowym.
- **Shadery** — maksymalnie trzy warstwy wbudowane w odtwarzacz.
- **Pobierz .p37** — tylko w internecie. Wyświetla pozycję pobierania w menu pauzy.
- **Ikona** — kafelek z arkusza sprite'ów: arkusz, rozmiar, x, y. Albo przycięta okładka PNG.
- **Eksportuj** — wybierz folder. Menu wyświetla komunikaty „BUDOWA ZAPISANA" lub „BUDOWA NIEPOWODZONA".

Stwórz kompilację z włączonym internetem. Ekran startowy może anonimowo zapytać serwer, czy hash tej kompilacji jest znany. To sprawdzenie ma dwa powody. Chroni mnie, autora PIX37, przed kopiami narzędzia, które nie były moje, aby je rozdawać. Chroni również osobę, która stworzyła grę. Przypisywanie gry innej osobie i wystawianie jej jako swojej staje się coraz powszechniejsze na rynku telefonów, takich jak Play (Android) lub iOS. Gra nadal się uruchamia. Jeśli komputer jest offline, intro informuje, że gra nie jest połączona. Ta linijka jest również wskazówką dla gracza, że kopia może nie pochodzić z legalnego źródła. To małe sprawdzenie, dzięki któremu jeśli twoja gra została skradziona i sprzedana bez twojej zgody przez kogoś innego, możesz zgłosić to w sklepie takim jak Play (Android), Steam lub itch.io. Serwer przechowuje czas pierwszego zarejestrowania hasha obok hasha i nie zmienia go później. Wcześniejszy czas to oryginalna kompilacja. Hash zarejestrowany później to późniejsza kopia.

Chcę, żeby ta kontrola chroniła ludzi, którzy tworzą gry. Silnik i tak już wykonuje za nich sporo pracy, a to jest jedna z takich rzeczy. Rozumiem, że może to budzić kontrowersje. Jeśli społeczność zdecyduje, że lepiej zrezygnować z tego mikro-DRM, to tak zrobię.

Stara kaseta z poprzedniego edytora może nie uruchomić się po zmianie języka. Otwórz projekt `.pix37` w tym edytorze i wyeksportuj nowy projekt `.p37`.

Połącz się z PIX37.PARTY

POŁĄCZENIE Z PIX37.PARTY znajduje się w aplikacji. Przeglądarka nie posiada strony logowania. Logujesz się za pomocą biletu sesji Steam lub za pomocą itch.io, używając pada i myszki. Usługa wyświetla unikalną nazwę użytkownika, składającą się z 2 do 16 znaków. Klucz licencyjny i identyfikator Steam nie są widoczne dla innych graczy.

Po zapoznaniu się z warunkami usługi, centrum stanowią Ulubione, Katalog, Ranking, Gracze i Twój profil.

Licencja programisty to edytor. Tworzysz grę, publikujesz ją i zostawiasz gwiazdki. Licencja gracza to gracz. Serca pochodzą od osób grających w Runnera.

- **Katalog** — przeglądaj gry. Filtry obejmują gatunek (maksymalnie trzy), rozmiar ekranu, rozmiar kasety oraz to, czy przesyłany plik zawiera projekt `.pix37`.
- **Ulubione** — gry, które zaznaczyłeś.
- **Prześlij** — wymagana jest licencja programisty. Wysyłasz plik `.p37` i możesz dołączyć projekt `.pix37`. Plik `.p37` to spakowany kod bajtowy. Jest trudny do odczytania i może być celowo zaciemniany. Plik `.pix37` to plik projektu, nad którym faktycznie pracujesz: źródło, obrazy i dźwięk, w formie, którą inna osoba może otworzyć i z której może się uczyć. Wysyłanie go pozwala innym twórcom zapoznać się z grą zamiast ze skompresowaną kasetą. Zawiera tytuł, autora, gatunki, okładkę, ikonę i nazwy osiągnięć. Przesłana liczba pamięci RAM to szczytowy poziom, jaki osiągnął kaseta.
- **Komentarze i osiągnięcia** — na stronie gry. Odblokowanie osiągnięcia po zalogowaniu jest zapisywane zarówno na tym koncie, jak i na komputerze.
- **Ranking** — twórcy, posortowani według gwiazdek lub serduszek. Każdy wiersz pokazuje obie wartości. Otwórz nazwę, aby zobaczyć gry tego autora, posortowane według serduszek.
- **Gracze** — konta uporządkowane według odblokowanych osiągnięć.

Kod

Język to niewielki skrypt w stylu Lua. Ta część zaczyna się od pustego projektu, uczy języka, a następnie pokazuje krótki przykład każdej funkcji.

Pierwszy program

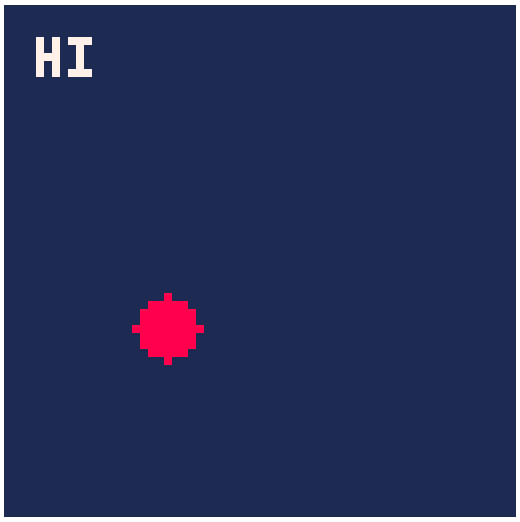
Utwórz projekt w rozdzielczości 64×64, 4-bitowym kolorze. Otwórz CODE. Zastąp zakładkę:

```
_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 uruchamia grę. Strzałki przesuwają kółko. Escape zatrzymuje grę i wraca do edytora.



Te same trzy wywołania: `cls(1)` maluje granatowy ekran, `print` rysuje HI, a `circfill` rysuje różowy okrąg.

Co i kiedy działa

- `_init` uruchamia się raz.
- `_update` działa z częstotliwością 60 Hz. Umieść tutaj ruch i przyciski.
- `_draw` maluje. Bez argumentu fps, maluje z szybkością, na jaką pozwala komputer i monitor. `_draw(30)` ogranicza malowanie do 30 fps i pozostawia logikę na 60.
- `_set_fps(n)` zmienia ten limit później. 0 go usuwa.

Nazwy nie są rozróżniane wielkością liter; edytor zapisuje je małymi literami. `function` i `fun` to to samo słowo kluczowe.

Język

Podstawy programowania

Program to lista wierszy odtwarzanych przez kasetę. Każdy nagłówek pokazuje najpierw kształt wiersza, a następnie krótki przykład. Wiersz zaczynający się od `--` to komentarz: notatka dla Ciebie. Gra go pomija.

Zmienne

Zmienna to nazwa, która przechowuje wartość. Wybierasz nazwę. Późniejsze wiersze mogą ją odczytać i zapisać nową wartość pod tą samą nazwą.

Kształt to `name = value`. `local name = value` zachowuje tę nazwę wewnątrz bieżącego bloku, więc reszta kasety jej nie widzi. Bez `local` nazwa jest globalna i widoczna dla każdej zakładki.

```
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` oznacza brak wartości. `true` i `false` oznaczają „tak” i „nie”. Nazwa, której nigdy nie ustawiłeś, będzie czytana jako `nil`.

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

Tabele

Tabela zawiera wiele wartości pod jedną nazwą. Lista używa liczb. Pierwszym elementem jest 1, a nie 0. Rekord używa nazw. Tabela może zawierać inną tabelę.

Kształt listy: `{ value, value }`. Kształt rekordu: `{ name = value, name = value }`. Lista rekordów to tabela, której elementami są tabele.

```
-- 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
```

Stan i ciało

Warunek to pytanie z odpowiedzią „tak” lub „nie”, na przykład `hp > 0` lub `btnd(4)`. Treść to grupa linii uruchamianych po wejściu do bloku. `if` i każda pętla używają tego samego schematu: najpierw warunek, potem treść, a potem `end`. `repeat` kończy się kodem `until` zamiast `end`.

Jeśli

`if` uruchamia swoje ciało tylko wtedy, gdy warunek jest prawdziwy. `elseif` próbuje innego warunku. `else` uruchamia się, gdy wszystkie warunki są fałszywe.

Kształt: `if condition then`, następnie korpus, następnie `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
```

Pętle

Pętla powtarza treść pętli. `while` najpierw sprawdza warunek i kontynuuje działanie, dopóki jest on spełniony. `repeat` uruchamia treść pętli, a następnie ją sprawdza, dzięki czemu treść pętli zawsze zostanie wykonana co najmniej raz. `for` zlicza. Ostatnia liczba jest uwzględniana. Bez trzeciej liczby krok wynosi 1, więc jeśli początek już minął koniec pętli, treść pętli nie zostanie wykonana. Odliczanie w dół wymaga kroku ujemnego.

```
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
```

Chodzenie po stole

`all` sprawdza każdy element listy, od pierwszego. `pairs` sprawdza każdą nazwę i wartość zapisaną pod tą nazwą.

```
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
```

Funkcje

Funkcja to nazwany przepis. Najpierw kształt `function name(arguments)`, potem ciało, a na końcu `end`. Wywołuje się ją, wpisując nazwę i nawiasy. `return` zwraca wartość i może wysłać więcej niż jedną. Argumenty są zmiennymi lokalnymi tej funkcji.

```
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` to funkcja, która widzi zmienne lokalne wokół siebie. To jest domknięcie. Tabela może je przechowywać i wywołuje się ją kropką.

```
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
```

Tak, nie i wybór

`not` odwraca warunek. `and` jest prawdą tylko wtedy, gdy obie strony są prawdziwe. `or` jest prawdą, gdy jedna ze stron jest prawdziwa. `a and b or c` wybiera `b`, gdy `a` jest prawdą, w przeciwnym razie `c`. `cond ? a : b` to ta sama idea w jednym wierszu: `a`, gdy warunek jest prawdziwy, `b`, gdy nie jest.

```
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)
```

Matematyka i tekst

`+ - * /` to dodawanie, odejmowanie, mnożenie i dzielenie. `//` dzieli i zachowuje liczbę całkowitą. `%` to reszta. `^` to potęgowanie. `..` skleja dwa teksty. `#t` to liczba elementów na liście. Porównania to `==` (to samo), `~=` lub `!=` (różne) i `< > <= >=`. `x += 1` dodaje 1 do x. Ten sam schemat działa dla `-= *= /= ^= %=`.

```
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
```

Gdzie mieszka miejscowy

`local` działa do momentu, aż `end` zamknie swój blok. `if` / `end` to osobny blok. Możesz podać lokalny kod nad wierszem, który go ustawia, ale do momentu uruchomienia tego wiersza wartością jest `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
```

Skoki

`::label::` oznacza linię. `goto label` przeskakuje tam, ale tylko wewnątrz tej samej funkcji.

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

Pamięć, krótko mówiąc

`peek` i `poke` analizują piksele na żywo za pomocą jednego 16-bitowego adresu. Nie przechowują drugiej kopii kasety. Adresy od `0x5F00` do `0x5FFF` to rejestry: kamery, klipu (tylko do odczytu), koloru rysowania, wzoru wypełnienia, trzech remapowań palety i numerów stron. Bajty po obu stronach tej dziury, w liczbie 65280, to okno pikseli.

Gdy ekran mieści się w tych 65280 bajtach (64×64 i 128×128 mieszczą się, a 256×256 nie), ekran rozpoczyna się od adresu 0, wiersz po wierszu. Pozostaje arkusz sprite'ów wybrany przez `0x5F41` (0 to arkusz 0). Jeśli arkusz jest większy niż pozostała przestrzeń, `0x5F42` wybiera stronę, a mapa nie pojawia się w oknie. Jeśli arkusz się mieści, ogon mapy jest mapowany z komórek mapy wybranej przez `0x5F43`, po jednym lub dwóch bajtach każda, a `0x5F44` stronicuje mapę, gdy nie mieści się ona w ogonie.

Gdy ekran się nie mieści, całe okno zajmuje jedną stronę o rozmiarze 65280 pikseli. `poke(0x5F40, n)` zmienia stronę ekranu. `poke(0x5F41, n)` z n różnym od 0 wyświetla zamiast niego arkusz n, a arkusz 0 nie ma widoku w tym trybie. `poke(0x5F43, n)` z n różnym od 0 wyświetla mapę n, co ma pierwszeństwo przed arkuszem. Mapa 0 w tym trybie to tylko `mget` i `mset`.

Kod `peek`, który nie trafia na piksel ani rejestr, zwraca 0. Kod `poke` w tym miejscu nic nie robi. Flagi sprite'ów, efekty specjalne i tabele danych nie znajdują się w tym oknie. Użyj kodów `fget`, `sfxget` i `dget`.

Praktyczne zastosowanie

Każda funkcja jest wymieniona na dole tej strony, wraz z praktycznymi zastosowaniami funkcji PIX37. Linki pod każdym przykładem otwierają dodatkowe informacje o funkcjach.



Płynny ruch

Wózek o wymiarach 128×128, widziany z góry. Sprite 1 jest już narysowany. Strzałki przyspieszają w dowolnym kierunku, a tarcie spowalnia. Sprite się zawija: połowa, która wychodzi z dołu, jest rysowana na górze, i tak samo jest z pozostałymi krawędziami.

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



Sprawdzanie, czy następny ruch jest zablokowany

Wózek 64×64. Podłoga i prawa ściana są już na mapie, pomalowane sprite'em 3, a flaga 0 jest ustawiona na tym sprite'ie. Strzałki przesuwają sprite'a 1. Słowo u góry oznacza „zablokowany”, gdy kolejny krok nachodziłby na pełny kafelek, i „wolny”, gdy tak się nie działo.

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



Chusteczka z klapką

Naciśnij A. Czyszczenie jest rysowane przez flip(), jeden czarny wiersz na raz, a nie wewnątrz _draw(). _draw nie przeszkadza podczas działania tej funkcji, więc każdy wiersz pozostaje na ekranie.

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



Tabela funkcji (klasa)

Sprite 1 jest już narysowany. make_monster działa jak klasa. Zwraca tabelę, a metody update i draw przechowują wartości x i y danego potwora. Liczba na górze wskazuje odległość, jaką pokonał.

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



Cztery sposoby odczytania przycisku

A to przycisk 4. Przytrzymaj go, a ładunek wzrośnie. Puść, a ładunek wróci do 0. Powtarzaj w górę i w dół, przytrzymując go, aby wiersz menu się przesuwał. BTND to pierwsza klatka naciśnięcia, której ten wózek używa tylko jako początku tego przytrzymania.

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



Strzelaj myszką

Sprite 1 jest już narysowany na środku. Biała kropka to kursor. Kliknięcie lewym przyciskiem myszy wystrzeliwuje w jego kierunku piksel. Kółko zmienia prędkość, wydrukowane w rogu.

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



Wpisz nazwę i potwierdź ją

Naciśnij A, aby otworzyć pole. Klawiszami w górę i w dół zmieniaj migającą literę. A wpisuje ją, X usuwa, Y to spacja, a B potwierdza. Następnie w wierszu pojawia się Twoje imię.

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



Kamera na odtwarzaczu

Wózek 128×128. Sprite 1 zaczyna się na środku podłogi z kafelków. Strzałki się poruszają. Kamera utrzymuje sprite'a w centrum. Współrzędne x i y w rogu nie przewijają się, ponieważ funkcja camera() bez argumentów przywraca widok dla HUD-a.

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



Kilka linijek powieści wizualnej

Linie są w formacie DATA, po angielsku i polsku. A kontynuuje lub wybiera opcję „zostań”, gdy opcja jest wyświetlana na ekranie. B wybiera opcję „wyjdź” po wybraniu opcji, a w przeciwnym razie zmienia język. X zmienia język, gdy opcja jest wyświetlana. Dialog używa czcionki 8 px, aby wyświetlać polskie litery.

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



Maluj metodą „peek and poke”

Wózek 64×64, więc adres 0 to piksel w lewym górnym rogu, a następny wiersz zaczyna się od 64. Wózek zapisuje piksele, kopiuje wiersz, czyści wiersz i drukuje odczytane wartości.

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



Lista, do której możesz dodać

A dodaje liczbę, do 8. B usuwa ostatnią. X usuwa pierwszą. Lista jest wyświetlana w dół ekranu.

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



Rozebrana nazwa

Wózek rozdziela ada,12,blue i drukuje nazwę, jej pierwszą literę, 12, oraz słowo blue.

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



Zagraj nutę, a następnie ją zmień

Wzór 0 ma długość jednej nuty, a każde powtórzenie ma tę samą głośność. A odtwarza go ponownie po zakończeniu tej nuty. B podnosi wysokość dźwięku i odtwarza tę nutę.

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



Piosenka zapisana w wózku

Dwa wzory zapętlają się w wierszu 0, po jednym na każdym z pierwszych dwóch kanałów. Wiersz pozostaje 0. Ch0 i ch1 to nuta, na której znajduje się każdy kanał. Edycja wzoru, LEN i LOOP w kasecie to sposób na sprawdzenie, jak te elementy sterujące działają w panelu muzycznym.

`songget songset music musicstat cls print _init _draw`



Trzy potwory w DATA

Panel DANE ma już kolumnę z nazwą i kolumnę z punktami zdrowia, ze śluzem, nietoperzem i okiem. A rani śluz. Liczby po prawej stronie to jego punkty zdrowia.

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



Zapamiętaj wynik

A dodaje punkt i go zapisuje. Wychodzi i otwiera kasetę ponownie: liczba nadal tam jest. B o niej zapomina.

`load btnd save delsave cls print _init _update _draw`




Drugi rozdział w innym pliku

Dwa wózki dzielą zadanie zapisu. Część 1: A oznacza zakończenie gry. Część 2, otwierana jako pierwsza, nakazuje dokończenie części 1. Następnie otwiera się część 2, a B kasuje zapis, aby można było ponownie rozegrać część 1.

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



Jedno osiągnięcie

Sprite 8 jest już narysowany. Naciśnij A dziesięć razy. Toast nadaje nazwę osiągnięciu, a sprite 8 pozostaje, ponieważ jest odblokowany. B usuwa to osiągnięcie z pliku lokalnego i resetuje licznik.

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



Pokój, do którego można wejść

Wózek o wymiarach 256×256. Biała kropka to kursor. Menu to „host”, „join” lub „server list”. Host otwiera pokój o nazwie „walk” i rozpoczyna grę. Przycisk „join” przewija adres (cyfry, kropka, dwukropek), a przycisk

„B” łączy. Lista serwerów przesuwa się w górę i w dół, a przycisk „A” dołącza do wiersza. W grze strzałki wskazują sprite'a 1. Jedna moneta jest wspólna: host ją przesuwa, a jej podniesienie dodaje 1 do wyniku danego gracza i tworzy kolejnego. Każdy sprite, w tym twój, ma obok siebie ten wynik. W rogu ekranu wyświetlana jest liczba graczy, adres LAN i publiczny adres hosta. Przycisk „B” wychodzi: ty i pozostali wracacie do menu, a kolejny host rozpoczyna nowy pokój z wynikiem 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, który czeka

A pokazuje następny wiersz. Funkcja zatrzymuje się po każdym zdarzeniu, aż do ponownego naciśnięcia A.

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



Poczekaj pół sekundy

Słowo zmienia się z ... na „hi” po około pół sekundy. Ekran nadal rysuje, podczas gdy coroutine jest wstrzymany.

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



Kręgi, które przybywają jedno po drugim

Pięć okręgów spływa z góry, w odstępach dwóch klatek, a następnie spada.

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



Dwie klatki, z zegara

Sprite 1 i sprite 2 to dwie klatki tej samej postaci. Obraz zmienia się cztery razy na sekundę.

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

Każda funkcja

Argumenty w nawiasach kwadratowych są opcjonalne. Indeksy tabel i ciągów znaków zaczynają się od 1, z wyjątkiem wierszy i kolumn DATA, które zaczynają się od 0. Kolory są indeksami palety.

Indeks

`_rysuj`
`_inicj`
`_mp`
`_mp_fail`
`_mp_dołącz`
`_mp_opuść`
`_mp_aktualizacja`
`_set_fps`
`_aktualizacja`

abs
jasne
achdef
osiągnąć
osiągnięty
dodaj
wszystko
twierdzić
atan2
zespół
nie
bor
btn
btnd
btnp
btnu
bxor
aparat
sufit
chr
ok
obwód
kliknij
kliknij
kliker
klip
kl
współtworzyć
kolor
podstawowe
bo
koszt
liczyć
kursor
dcols
del
delikatesy
usuńzapisz
dget
śpi
dset
dstr
fget
wypełnienie
odwróć
flr
dotrzeć
fset
informacje
ipary
wyłączenie
klucz
klucz
klucz
linia
załadować
log
lszr
mapa
maks
memcpy
memset
mget
środek
min
mysz
mset
muzyka
Musicstat
ogłosić
zadzwoń
następny
nhost
nie dołącz
nkb
nkbstr
nie
wyjdź
nlista
nie słuchaj
uwaga
niegracze
nie wyślij
niestun
zamów
owalny
wypełnienie owalne
pakować
pary
kolego
palrgb
blady
pcall
zerknij
zerknij2
zerknij 4
wyk
strona
szturchać
szturchać2
szturchać 4
wydrukować
wydrukuj 10
drukuj8
pset
prosto
wypełnienie prostownicze
koniec
rotl
rotr
prawda
prawidłowe wypełnienie
zapisz
miejsce zapisu
ekran
przewiń
wybierz
mapa ustawień
zestaw palet
zestawypr
settex
zwrot
sfx

sfxget	grzech	statystyka
sfxhdr	śpiewać	pod
zestaw sfx	zestaw piosenek	czas
sfxstat	podzielić	linia
sget	wiosna	tonum
znak	kwadrat	tostr
shl	srand	rozpakować
statystyki pokazowe	zestaw	whoami
shr	sspr	plon

Cykl życia

_init

`_init() ... end`

Uruchamia się jednokrotnie podczas startu kasety. Kod znajdujący się poza jakimkolwiek hakiem jest również składany do _init.

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

_update

`_update() ... end`

Logika gry. Wywoływana z częstotliwością 60 Hz, nawet gdy _draw jest ograniczony lub host jest wolniejszy. Przeczytaj przyciski tutaj.

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

_draw

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

Rysuje jedną klatkę. Bez argumentu lub zerowej wartości, uruchamia się z częstotliwością wskazywaną przez hosta. Przekaż n (1..999), aby ograniczyć rysowanie do n klatek na sekundę. _update utrzymuje się na poziomie 60 Hz.

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

_set_fps

```
_set_fps(n)
```

Zmienia limit _draw podczas działania gry. 0 oznacza brak limitu. n jest ograniczone do wartości 0..999.

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

flip

```
flip()
```

Wyświetla ekran i ustępuje miejsca następnej klatce. Używaj go tylko wtedy, gdy rysujesz własną pętlę klatek. Nie wykonuj funkcji „busy-wait” po jej zakończeniu.

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

time

```
time()
```

Sekundy od _init.

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

Wejście

btn

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

1, gdy wciśnięty jest przycisk b, w przeciwnym razie 0. p to gracz, 0..7. Bez argumentów zwraca maskę bitową gracza 0. Przyciski: 0 w lewo, 1 w prawo, 2 w górę, 3 w dół, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 wybierz, 11 start.

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

btnd

`btnd(b, [p])`

Prawda tylko w ramce _update, gdy przycisk b zostanie naciśnięty.

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

btnp

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

Prawda po naciśnięciu, potem ponownie po przytrzymaniu: 30 taktów opóźnienia, potem co 8 taktów. Bez przycisku zwraca maskę.

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

btnu

`btnu(b, [p])`

Prawda tylko w momencie zwolnienia przycisku na ramce.

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

mouse

```
mouse()
```

Zwraca współrzędne myszy x, y w pikselach gry.

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

clickl

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

Lewy przycisk myszy. Przytrzymano przycisk clickl(). Kliknięcie clickl(1) oznacza naciśnięcie krawędzi.

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

clickr

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

Prawy przycisk myszy. Ta sama krawędź jednoargumentowa co w przypadku kliknięcia.

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

clickm

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

Środkowy przycisk lub kliknięcie kółka przewijania.

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

scroll

`scroll()`

Ruch kółka myszy w tej klatce. Wartość dodatnia i ujemna zależy od kierunku obrotu kółka.

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

keyon

`keyon()`

Otwiera pole tekstowe sterowane przyciskami. Klawisze w górę i w dół zmieniają litery od A do Z. A wpisuje tę literę. X usuwa. Y to spacja. B potwierdza i zamyka pole. Klawiatura nie wpisuje tekstu.

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

keyoff

`keyoff()`

Ukrywa pole tekstowe.

```
keyoff()
```

keystr

`keystr([s])`

Tekst w polu, następnie 1 w ramce B potwierdza, po czym następuje zmiana litery w górę i w dół. Ta litera jest pusta po zamknięciu pola. Klawisz keystr("") czyści tekst.

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

keystat

```
keystat()
```

1, gdy pole tekstowe jest otwarte, w przeciwnym wypadku 0.

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

Rysunek

screen

```
screen()
```

Aktualny rozmiar ekranu kasety w pikselach. Zwraca szerokość, a następnie wysokość. To rozdzielczość gry, a nie okna edytora.

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

cls

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

Czyści ekran. Domyślny kolor to 0.

```
cls(1)
```

color

```
color(col)
```

Ustawia kolor rysowania używany, gdy późniejsze wywołanie pominie kolumnę. Bez argumentu zwraca bieżący kolor.

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

camera

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

Przesuwa każde wywołanie rysowania. camera() resetuje się do 0, 0.

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

clip

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

Ogranicza rysowanie do prostokąta. Piąty argument różny od zera przecina bieżący klip. Funkcja clip() przywraca pełny ekran. Rejestry klipu są dostępne tylko do odczytu.

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

cursor

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

Ustawia miejsce, w którym zostanie wykonany następny wydruk bez współrzędnych, a opcjonalnie także kolor. cursor() zwraca x, y.

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

pset

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

Ustawia jeden piksel ekranowy. Wlicza się do statystyk pikseli.

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

pget

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

Kolor piksela ekranu w punktach x, y.

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

print

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

Rysuje tekst 3×5 (UTF-8, wykadrowany z czcionki Fusion). Bez x, y tekst jest drukowany w miejscu kursora i przechodzi do następnego wiersza.

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

log

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

Dodaje argumenty, oddzielone spacjami, do bufora tekstowego. pix37_run drukuje ten bufor na standardowym wyjściu. Nie rysuje.

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

print8

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

Tak samo jak w wersji drukowanej, ale z czcionką Fusion o rozmiarze 8 px.

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

print10

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

Tak samo jak w wersji drukowanej, ale z czcionką Fusion o rozmiarze 10 px.

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

line

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

Linia od x0,y0 do x1,y1. Dwa argumenty są kontynuacją poprzedniego końca. Funkcja line() resetuje ten łańcuch.

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

circ

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

Zarys okręgu. Promień podany jest w pikselach.

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

circfill

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

Wypełniony okrąg.

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

oval

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

Kontur elipsy wewnątrz pola ograniczającego. Narożniki są wliczone.

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

ovalfill

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

Wypełniona elipsa w polu ograniczającym.

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

rect

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

Zarys prostokąta. Narożniki są inkluzywne.

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

rectfill

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

Wypełniony prostokąt.

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

rrect

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

Zaokrąglony kontur prostokąta. x, y to lewy górny róg. w i h to rozmiar w pikselach; oba muszą być większe od 0. r to promień narożnika, domyślnie równy 0. Jest on przycięty do połowy krótszego boku.

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

rrectfill

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

Wypełniony zaokrąglony prostokąt. Te same argumenty co rrect.

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

fillp

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

16-bitowy wzór wypełnienia dla później wypełnianych kształtów. 0 oznacza wypełnienie. Funkcja fillp() zwraca bieżący wzór. Bity przechodzą przez wzór 4×4.

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

pal

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

Przemapowuje kolor c0 na c1. Tryb 0 to paleta rysowania, 1 to paleta wyświetlania, 2 to paleta wzoru wypełnienia. pal() resetuje.

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

palrgb

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

Ustawia jeden wpis palety na RGB, każdy kanał 0..255.

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

palt

```
palt([col, t])
```

Przezroczystość. palt(col) lub palt(col, true) sprawia, że kolor jest przezroczysty. palt(col, false) sprawia, że jest pełny. palt() sprawia, że każdy kolor jest nieprzezroczysty.

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

setpalette

```
setpalette(n)
```

Wybiera bank palet nr. setpalette() zwraca bieżący bank.

```
setpalette(1)
```

tline

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

Rysuje linię na ekranie, jednocześnie pobierając próbki pikseli mapy z pozycji mapy mx, my. mdx, mdy przesuwają próbkę mapy wzdłuż linii.

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

Sprite'y i mapy

spr

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

Rysuje sprite'a n w współrzędnych x, y. w i h to rozmiar w pikselach, domyślnie 8. fx i fy odwracają się poziomo i pionowo, gdy wartość jest różna od zera. tex to zakładka arkusza sprite'ów. rot to kąt w obrotach (0..1), ten sam kierunek co sin i cos. 0 pozostawia sprite'a nieobróconego.

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

sspr

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

Kopiuje prostokąt arkusza sprite'ów na ekran, skalowany do dw przez dh. Domyślnie zachowany jest rozmiar źródłowy.

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

sget

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

Kolor jednego teksela na arkuszu sprite'ów.

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

sset

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

Zapisuje jeden teksel na arkuszu sprite'ów.

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

fget

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

Flagi sprite'ów. fget(n) zwraca bajt flagi. fget(n, f) ma wartość true, gdy ustawiony jest bit flagi f (0..7).

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

fset

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

Ustawia flagi sprite'ów. fset(n, v) zapisuje cały bajt. fset(n, f, v) ustawia jeden bit.

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

setspr

```
setspr(n)
```

Wybiera aktywny arkusz sprite'ów. setspr() zwraca bieżący arkusz. Ten sam bank co settex.

```
setspr(1)
```

settex

```
settex(n)
```

Alias setspr.

```
settex(0)
```

map

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

Rysuje mapę. Komórka cx, cy mapy jest rysowana na ekranie sx, sy. cw, ch równe 0 oznacza całą mapę. Warstwa jest akceptowana i ignorowana. tex jest arkuszem zestawu kafelków.

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

mget

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

Identyfikator sprite'a w komórce mapy cx, cy.

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

mset

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

Zapisuje sprite'a n do komórki mapy cx, cy.

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

setmap

```
setmap(n)
```

Wybiera mapę n. setmap() zwraca bieżącą mapę. Mapa 0 jest mapą rzeczywistą.

```
setmap(1)
```

Pamięć

peek

```
peek(addr)
```

Odczytuje jeden bajt z okna o rozmiarze 64 KB. Okno to ekran na żywo, następnie piksele sprite'ów, a następnie komórki mapowania, gdy ekran mieści się w 65280 bajtach. Większe ekrany są stronicowane za pomocą poke(0x5F40, strona). Adresy, które nie są mapowane, zwracają 0. Zapisuje się również @addr.

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

poke

```
poke(addr, ...)
```

Zapisuje bajty pod adresem. Dodatkowe argumenty zapisują kolejne bajty. Niezmapowane adresy są ignorowane. Zapisuje również $addr, val. Dotknięcie ekranu liczy się jako zapis piksela.

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

peek2

```
peek2(addr)
```

Wartość 16-bitowa bez znaku, little-endian, pod adresem.

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

poke2

```
poke2(addr, val)
```

Zapisuje 16-bitową wartość bez znaku w formacie little-endian.

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

peek4

`peek4(addr)`

Wartość 32-bitowa bez znaku, w formacie little-endian.

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

poke4

`poke4(addr, val)`

Zapisuje 32-bitową wartość bez znaku, w formacie little-endian.

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

memcpy

`memcpy(dst, src, len)`

Kopiuje len bajtów w obrębie tego samego okna o rozmiarze 64 KB.

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

memset

`memset(dest, val, len)`

Wypełnia len bajtów zaczynając od dest wartością val.

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

Matematyka

abs

`abs(x)`

Wartość bezwzględna.

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

flr

`flr(x)`

Zaokrągla w kierunku minus nieskończoności.

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

ceil

`ceil(x)`

Zaokrągla w kierunku plus nieskończoności.

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

max

`max(a, b)`

Większa z dwóch wartości.

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

min

`min(a, b)`

Mniejsza z dwóch wartości.

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

mid

`mid(a, b, c)`

Środkowa z trzech wartości. Przydatna jako zacisk.

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

sgn

`sgn(x)`

Znak x: -1, 0 lub 1.

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

sqrt

`sqrt(x)`

Pierwiastek kwadratowy. Argument ujemny nie jest liczbą.

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

sin

`sin(x)`

Sinus. x jest w obrotach: 0, 0,25, 0,5, 0,75, okrąża okrąg raz. Nie w radianach.

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

cos

`cos(x)`

Cosinus z kolei stosuje tę samą konwencję co sinus.

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

atan2

`atan2(dx, dy)`

Kąt wektora dx, dy, w obrotach od 0 do 1.

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

rnd

`rnd([x])`

Liczba losowa w zakresie [0, x]. Domyślnie x wynosi 1, więc rnd() wynosi 0..1. W przypadku kostki do gry: flr(rnd(6))+1.

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

srand

`srand(seed)`

Uruchamia generator losowy tak, aby późniejsza sekwencja mogła zostać powtórzona.

```
srand(1)
```

band

`band(a, b)`

AND bitowy. Zapisuje się go również jako a i b.

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

bor

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

LUB bitowy. Zapisuje się go również jako | b.

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

bxor

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

Operacja XOR bitowa. Zapisuje się ją również jako ^^ b.

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

bnot

```
bnot(a)
```

Bitowe NOT. Zapisywane również jako ~a.

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

shl

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

Przesunięcie w lewo. Zapisuje się je również jako << n.

shr

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

Przesunięcie arytmetyczne w prawo. Zapisywane również jako >> n.

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

lshr

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

Przesunięcie logiczne w prawo. Zapisano również jako >>> n.

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

rotl

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

Obróć bity w lewo. Zapisuje się je również jako <<> n.

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

rotr

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

Obróć bity w prawo. Zapisywany również jako >>< n.

Tabele i wartości

add

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

Dołącza v do tabeli t. Jeśli podano i, wstawia pod tym indeksem od 1.

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

del

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

Usuwa pierwszy wpis równy v.

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

deli

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

Usuwa indeks i i zwraca tę wartość. W przypadku braku indeksu i usuwa ostatni wpis.

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

all

```
all(t)
```

Iterator po części tablicy, w kolejności. Dziury zatrzymują przejście w taki sam sposób, w jaki kończy się tablica oparta na 1.

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

foreach

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

Wywołuje fn(v) dla każdego elementu tablicy.

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

pairs

```
pairs(t)
```

Iterator po każdym kluczu i wartości, włączając w to pola nazwane.

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

ipairs

```
ipairs(t)
```

Iterator po części tablicy z indeksem zaczynającym się od 1.

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

next

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

Zwraca następny klucz i wartość po k. next(t) zaczyna się od początku. Używane przez pary.

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

count

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

Długość części tablicy. Jeśli podano v, zlicza, ile wpisów jest równych v.

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

pack

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

Pakuje argumenty do nowej tabeli.

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

unpack

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

Zwraca wartości od t[i] do t[j] jako osobne wartości. Wartości domyślne obejmują całą tablicę.

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

select

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

select("#", ...) zwraca liczbę następujących argumentów. select(i, ...) zwraca argumenty od i w górę.

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

assert

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

Zatrzymuje się z błędem, gdy c ma wartość false lub nil. msg zastępuje domyślny tekst.

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

pcall

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

Wywołuje f. W przypadku powodzenia zwraca wartość true i wyniki. W przypadku błędu zwraca wartość false i tekst błędu.

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

Smyczki

chr

```
chr(n)
```

Jeden znak z kodu bajtowego 0..255.

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

ord

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

Kody bajtów s od indeksu i do j. Indeksy zaczynają się od 1. Domyślnie jest to pierwszy znak.

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

sub

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

Podciąg od i do j, oparty na 1. Pomiń j, aby wziąć ogon.

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

split

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

Dzieli s na tabelę. Domyślnie sep to przecinek. Liczba sep to szerokość fragmentu. conv konwertuje fragmenty z tonum, gdy ma wartość true.

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

tonum

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

Analizuje liczbę. Podstawą może być 2..36. Zwraca wartość nil, jeśli tekst nie jest liczbą.

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

tostr

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

Zamienia wartość na tekst. digits ustawia miejsca dziesiętne dla liczb.

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

Audio

sfx

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

Odtwarza wzorzec n i zwraca kanał. Pomiń parametr ch lub przekaż wartość -1, a wybrany zostanie pierwszy wolny kanał. Parametr ch -2 zatrzymuje wzorzec n na każdym kanale. Parametr n -1 zatrzymuje kanał lub wszystkie kanały, gdy parametr ch jest pominięty. Parametr n -2 pozwala zakończyć pętlę zamiast ją zapętlać. Funkcja sfx() bez argumentów zwraca wartość -1.

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

sfxget

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

Zwraca tabelę jednego kroku wzorca: [1] wysokość dźwięku, [2] fala, [3] głośność, [4] efekt. Wysokość dźwięku wynosi 0–63 (C-0 do D#5). Fala wynosi 0–15. Głośność wynosi 0–7, a 0 oznacza ciszę. Efekt wynosi 0–7.

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

sfxset

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

Zapisuje jeden krok wzorca. Wymagane jest podanie wszystkich sześciu argumentów.

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

sfxhdr

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

Nagłówek wzorca. Funkcja sfxhdr(n) zwraca tabelę nagłówków. Funkcja sfxhdr(n, klucz, wartość) ustawia jedno pole: prędkość, wejście pętli, wyjście pętli, poziom, szum, zmiażdżenie, dtune, pogłos, tłumienie, swing, kroki. Liczby 0–10 wybierają te same pola w tej kolejności. prędkość to TIC, liczba uderzeń na krok. Wyższa wartość oznacza wolniejszą wartość. 15 to 120 uderzeń na minutę. 16-bitowa wersja akceptuje wartości 1–255. Węższe syntezatory są przyciągane do listy TIC.

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

sfxstat

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

Identyfikator wzoru odtwarzany jest na kanale ch lub -1. Domyślny kanał to 0.

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

note

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

Odtwarza jedną nutę bez schematu. Wysokość dźwięku 0–63, fala 0–15, głośność 0–7, efekt 0–7. Liczba taktów określa czas trwania, w tych samych taktach 120 Hz, co TIC. Pomiń takty, a efekt trwa 15, czyli jeden domyślny krok. Efekt działa przez ten czas, po czym kanał wycisza się.

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

music

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

Uruchamia utwór n od początku. Funkcja music (-1) zatrzymuje się. Wywołanie funkcji music z identyfikatorem, który jest już odtwarzany, powoduje jej uruchomienie. Funkcja music() zwraca identyfikator odtwarzanego utworu lub wartość -1. Fade to milisekundy: utwór stopniowo się pojawia lub wycisza, gdy n wynosi -1. Ustawione bity maski to kanały używane w utworze. Wartość 0 oznacza wszystkie osiem. Pozostałe kanały pozostają wolne dla efektów specjalnych.

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

musicstat

```
musicstat()
```

Tabela odtwarzanego utworu. song i row to identyfikator utworu i wiersz lub -1, gdy nic nie jest odtwarzane. step i pos mają osiem liczb. step[1] to kanał 0, step[8] to kanał 7. step to indeks nuty lub -1, gdy kanał jest cichy. pos to odległość, jaka dzieli nutę od 0 do 1. tick to liczba tyknięć, które zostały już zagrane w danym wierszu. played to liczba wierszy rozpoczętych od funkcji music(), wliczając wiersz, który jest odtwarzany.

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

songget

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

Identyfikator wzorca przechowywany w komórce utworu. ch to 0..7. 255 oznacza, że komórka jest pusta. To jest wywołanie na żywo; tekst podpowiedzi do kodu nadal pokazuje starszą kolejność argumentów.

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

songset

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

Zapisuje identyfikator wzorca w komórce utworu. Użyj kodu 255, aby wyczyścić komórkę.

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

stat

```
stat(n)
```

Tylko dźwięk. 16..23 to identyfikator sfx na kanale n-16. 24..31 to wzór muzyczny na kanale n-24. Każda inna wartość n zwraca 0. Czas klatek i pamięć RAM to parametry wydajności i informacji, a nie statystyki.

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

Tabele danych

dget

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

Odczytuje komórkę tabeli danych t z panelu DANE. Wiersz i kolumna zaczynają się od 0.

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

dset

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

Zapisuje komórkę. Użyj ciągu znaków dla kolumny tekstowej.

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

dstr

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

Komórka jako tekst. Liczby są konwertowane.

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

drows

```
drows(t)
```

Ile wierszy ma tabela t.

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

dcols

```
dcols(t)
```

Ile kolumn ma tabela t.

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

Zapisz i osiągnięcia

save

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

Zapisuje v pod kluczem k w pliku zapisu. v może być liczbą, ciągiem znaków lub tabelą tych wartości. Plik jest skompresowany.

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

load

`load(k)`

Odczytuje klucz k. Zwraca nil, jeśli klucz nie występuje.

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

delsave

`delsave(k)`

Usuwa klucz k z zapisu.

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

saveslot

`saveslot([name])`

Wybiera plik zapisu w folderze autora. W przypadku braku nazwy, polem zapisu jest tytuł gry. Zwraca 1, gdy ten plik już istnieje.

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

achieve

`achieve(i)`

Odblokowuje osiągnięcie „i”, od 0 do 15. Przechowywane lokalnie. Kompilacja Steam ustawia również ACH_i. Sesja katalogowa zgłasza to na konto gracza.

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

achieved

```
achieved(i)
```

1 gdy osiągnięcie i jest już odblokowane.

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

achdef

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

Tekst toastu za osiągnięcie i, wyświetlany przez 15 sekund od dołu ekranu. spr to opcjonalny kafelek 8×8, tex jego arkusz.

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

achclear

```
achclear(i)
```

Czyści osiągnięcie „i” w pliku lokalnym. Ustawione wcześniej ACH_i Steam pozostaje ustawione.

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

Sieć

whoami

```
whoami()
```

Twój identyfikator gracza (0–15) w trakcie sesji. W trybie offline wynosi -1.

```
id = whoami()
```

nhost

`nhost()`

1, jeśli ta maszyna jest hostem, w przeciwnym wypadku 0.

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

nplayers

`nplayers()`

Ile aktywnych identyfikatorów istnieje w 0..15.

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

nlisten

`nlisten()`

Rozpoczyna hostowanie protokołu UDP i zwraca adres LAN jako ip:port.

```
addr = nlisten()
```

njoin

`njoin(addr)`

Łączy ip lub ip:port.

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

nleave

`nleave()`

Opuszcza sesję. Uczestnicy widzą _mp_leave z argumentem „0”.

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

nsend

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

Wysyła stratną wiadomość. Nazwa to zdarzenie. Następnie można podać maksymalnie cztery dodatkowe wartości. p od 0 do 15 szeptów do tego samego użytkownika; pomiń p, aby wysłać do wszystkich.

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

ncall

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

Taki sam kształt jak w przypadku nsend, ale wiadomość jest ponawiana aż do potwierdzenia. Trzy kolejne błędy powodują przerwanie sesji.

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

nlan

```
nlan()
```

Aktualny adres IP:port sieci LAN po nlisten.

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

nstun

```
nstun()
```

Twój publiczny adres IP:port ze STUN. Gość z innej sieci może spróbować się do niego podłączyć.

```
pub = nstun()
```

setturn

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

Funkcja TURN jest wyłączona, gdy ścieżka bezpośrednia zawiedzie. Funkcja setturn() z pustymi argumentami wyłącza ją. Adres to host:port.

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

nlist

`nlist(game)`

Tabela otwartych sesji dla tej nazwy gry. Każdy wpis zawiera adres, graczy i nazwę.

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

nannounce

`nannounce(game)`

Heartbeat, aby ten host pojawił się w nlist dla gry. Ustaw timer podczas hostowania.

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

nkb

`nkb([on])`

Pole adresu. nkb(1) je otwiera, nkb(0) ukrywa, nkb() informuje, czy jest otwarte. Przesuwając w górę i w dół, zmieniasz znak, 0–9, kropkę lub dwukropek. A zapisuje. X usuwa. Klawiatura nie wpisuje tekstu.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

Tekst w polu adresu, następnie znaki w górę i w dół ulegają zmianie. nkbstr("") czyści tekst.

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

_mp

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

Piszesz tę funkcję. Działa ona po otrzymaniu wiadomości. „from” to identyfikator nadawcy. „name” to zdarzenie. Następnie można podać maksymalnie cztery wartości.

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

_mp_join

`_mp_join(p)`

Piszesz tę funkcję. Odbiorca otrzymał identyfikator p. Funkcja ta jest również uruchamiana dla Ciebie; whoami() jest tym identyfikatorem.

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

_mp_leave

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

Piszesz tę funkcję. Peer p lewy. Dlaczego 0 jest normalnym odejściem. Dlaczego 1 jest utraconym połączeniem.

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

_mp_update

```
_mp_update()
```

Piszesz tę funkcję. Działa co 15 ramek _update, około 4 Hz. To dobre miejsce do nsendowania pozycji.

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

_mp_fail

```
_mp_fail()
```

Piszesz tę funkcję. Sesja jest martwa: host zniknął lub ncall nie powiódł się trzy razy.

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

Korutyny

cocreate

```
cocreate(f)
```

Tworzy współprogram z funkcji f. Nie jest uruchamiany do momentu coresume.

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

coresume

`coresume(c, ...)`

Uruchamia współprogram c, aż zwróci lub zwróci wartość. W przypadku powodzenia: true i zwrócone wartości. W przypadku błędu: false i błąd.

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

costatus

`costatus(c)`

„zawieszony”, „działający” lub „martwy”.

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

yield

`yield(...)`

Oddaje kontrolę osobie, która wywołała coresume. Wartości stają się wynikami tego coresume.

```
yield(1)
```

Mierzyć

info

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

Rysuje HUD na żywo w _draw: fps (średnia 1 sekunda), upd, drw, px, max, ram, peak, ramc i net, gdy sesja jest aktywna. info() używa kursora print. info(col) ustawia tylko kolor. info(x, y) lub info(x, y, col) ustawia go. Liczby używają B, k lub m, aby pasowały do małego ekranu. Samo wywołanie nie jest liczone w px.

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

showstats

`showstats([on])`

Ten sam HUD co info, rysowany automatycznie w punkcie 0, 0 po _draw, ignorując kamerę i klip. showstats(0) ukrywa go. showstats(1) lub showstats() go pokazuje.

```
showstats(1)
```

perf

`perf(n)`

Jedna liczba HUD, więc możesz ją narysować sam. 0 fps, 1 upd, 2 drw, 3 px, 4 max (najcięższa klatka w tej sekundzie), 5 ram, 6 ramc (ram plus spakowana kaseta), 7 szczytowa pamięć RAM od uruchomienia kasety.

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