Manuel PIX37

Éditeur, puis code. Copie hors ligne du livre sur pix37.party/docs.

Éditeur

PIX37 DEV est l'éditeur. Vous y dessinez, programmez et écoutez le jeu, puis vous exportez une cartouche ou une version autonome. Il permet également de jouer à une cartouche et d'ouvrir PIX37.PARTY pour télécharger des cartouches et consulter le catalogue. PIX37 DEV EDUCATIONAL est le même éditeur, mais il est gratuit. Il ne permet pas de créer de versions, d'exporter des cartouches et n'a pas accès à PIX37.PARTY. PIX37 Console est le lecteur. Il lit les cartouches `.p37` et vous permet d'ouvrir PIX37.PARTY pour télécharger des cartouches et parcourir le catalogue.

Projets et cartouches de jeu

Un fichier `.pix37` correspond au projet : fichiers sources, feuilles de sprites, cartes, sons et tables de données. Un fichier `.p37` correspond à la cartouche de jeu exécutée par le joueur : du bytecode compacté.

Les sauvegardes et les succès se trouvent dans le dossier de l'auteur, à côté des données de l'application. Sous Windows, cela correspond à `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` pour une sauvegarde et à `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` pour un succès. Sous Linux, les mêmes fichiers sont nommés `~/.local/share/PIX37/Mira/Pong.p37s` et `~/.local/share/PIX37/Mira/Pong.ach`. Ada est le compte utilisateur sur l'ordinateur. Mira est le nom de l'auteur, et ce nom correspond au dossier. Pong est l'emplacement de sauvegarde, qui par défaut correspond au titre du jeu. `save` crée le fichier `.p37s`. Le premier succès débloqué crée le fichier `.ach`. Si les fichiers sont déjà présents, le programme les lit au démarrage de la cartouche de jeu.

Menu développeur

La touche Échap ouvre le menu développeur. En cours de partie, elle permet de revenir à l'éditeur. Confirmez avec Entrée ou le bouton A (V par défaut). Le bouton Retour est B (C par défaut).

Sans projet chargé, le menu est le suivant : CRÉER UN PROJET, CHARGER UN PROJET, CONSTRUIRE, CHARGER UNE CARTE DE JEU, SE CONNECTER À PIX37.PARTY, PARAMÈTRES.

Une fois un projet chargé, la procédure est la suivante : ENREGISTRER LE PROJET, CHARGER LE PROJET, CRÉER UN PROJET, RESSOURCES, CONSTRUIRE, CHARGER LA CARTOUCHE DE JEU, SE CONNECTER À PIX37.PARTY, PARAMÈTRES.

Ctrl+S enregistre le projet ouvert. Ctrl+R l'exécute.

Créer un projet

L'outil CRÉER UN PROJET demande la cartouche de jeu avant de commencer la création du projet.

- **VIDÉO** — 1-BIT (2 couleurs), 2-BIT (4), 4-BIT (16), 8-BIT (256).
- **AUDIO** — profondeur d'échantillonnage du synthétiseur : 16, 8, 4, 2 ou 1 bit.
- **ÉCRAN** — Les préréglages incluent 64×64, 128×128, 160×90, 256×256 et 320×180. La taille PERSONNALISÉE vous permet de choisir n'importe quelle taille de 1×1 à 512×512.
- **PALETTES** — banques de couleurs. La première banque est initialement composée des 16 couleurs ci-dessous. CHARGER UNE PALETTE applique une palette intégrée : PICO-8 16, PICO-8 32, Game Boy, 1 bit, CGA, Sweetie-16, et une liste plus longue issue de Lospec (Resurrect 64, Apollo, Endesga 32 et 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century, etc.). Vous pouvez également charger un fichier de palette ou votre propre image PNG et extraire la palette de cette image.



Les index de couleur sont utilisés par `cls`, `pset`, `spr`, etc. Vous pouvez redessiner n'importe quel index dans l'éditeur de palette.

Les cinq panneaux

L'éditeur se compose des panneaux JEU, PEINTURE, CODE, SON et DONNÉES, dans cet ordre. La carte se trouve dans le panneau PEINTURE. Les effets sonores et les musiques se trouvent dans le panneau SON. Cliquez sur un panneau pour y travailler. Lorsque le panneau CODE est actif, les panneaux SON et DONNÉES ignorent les touches du clavier : la saisie au clavier ne déclenche donc pas la lecture du piano.



Disposition complète, écrite `1:1:4:1:1`. Chaque nombre indique la taille de la ligne occupée par le panneau : JEU, PEINTURE, CODE, SON, DONNÉES. Un petit écran est centré dans une cellule de 128 × 128 pixels. Un écran plus grand que 128 pixels élargit la colonne JEU.

PARAMÈTRES → GLOBAL → MISE EN PAGE permet de faire défiler les agencements :

Code	Ce que vous voyez
`1:1:4:1:1`	Tout. JEU et SON empilés, PEINTURE et DONNÉES empilées, CODE à droite.
`1:1:4:1:1 2X`	Même disposition. Le code est dessiné à une densité deux fois supérieure, donc deux fois plus de texte peut y tenir.
`1:0:1:0:1`	JEU, DONNÉES, CODE.
`1:0:1:0:1 2X`	Même disposition, avec le CODE plus dense.
`0:1:1:1:1`	Pas d'aperçu du jeu. PEINTURE, CODE, SON, DONNÉES.
`0:1:1:1:1 2X`	Même disposition, avec le CODE plus dense.
`0:1:4:1:2`	Studio. CODE large, une bande DATA plus haute, PEINTURE et SON.
`0:1:4:1:2 2X`	Même disposition, avec le CODE plus dense.

`0:0:4:0:4`	CODE et DONNÉES uniquement, les deux étant volumineux.
`0:0:4:0:4 2X`	Même disposition, avec le CODE plus dense.
`0:1:4:1:0`	PEINTURE, SON et un code étendu.
`0:1:4:1:0 2X`	Même disposition, avec le CODE plus dense.
`0:4:0:0:0`	Peinture uniquement, 256×256, un pixel de la feuille correspond à un pixel de la toile.

Une ligne `2X` conserve les mêmes panneaux. Le panneau CODE est dessiné à une résolution deux fois supérieure et réduit dans la même cellule ; ainsi, le texte, les tabulations et la gouttière sont deux fois moins hauts et deux fois plus de code source s'affiche à l'écran.

JEU

Aperçu de la cartouche de jeu en cours d'exécution. Cliquez sur le panneau JEU ou appuyez sur Ctrl+R pour lancer cet aperçu. Le panneau se configure dans PARAMÈTRES → PANNEAU JEU.

- **Aperçu** — couleur inactive, continu ou rechargement automatique lors de la modification.
- **Jouer dans une nouvelle fenêtre** — le jeu s'ouvre à côté de l'éditeur.
- **Affichage** — aperçu en RGB888 ou RGB565.
- **JEU 64** — un écran 64×64 est soit centré 1:1 dans la cellule, soit mis à l'échelle 2× pour la remplir.
- **Shaders** — CRT et filtres du jeu, distincts des filtres de l'éditeur.
- **Couleur** — arrière-plan du panneau.

PEINTURE

Des feuilles de sprites et, du côté carte du même panneau, des cartes en tuiles. Les feuilles peuvent être plus grandes que l'écran. Les tailles d'aperçu des sprites sont de 8, 16, 32, 64 et 128 (128 uniquement si la feuille fait au moins 128 px). Utilisez la molette de la souris pour zoomer sur l'aperçu des sprites. Les tailles de pinceau sont de 1, 2, 3 et 4.

Clé	Outil
B	Crayon
M	Sélection rectangulaire
Q	Lasso
S	Timbre

G	Seau
je	compte-gouttes
V	Se déplacer
L / Maj+L	Ligne / courbe
U / Maj+U	rectangle creux / ellipse creuse
F / Maj+F	Rectangle plein / ellipse pleine
D	Tremblement
[ ]	Brosse plus petite / plus grande
Ctrl+Z / Ctrl+Y	Annuler / Rétablir. Ctrl+Maj+Z rétablit également.
Ctrl+C / Ctrl+X / Ctrl+V	Copier, couper, coller
Ctrl+A	Sélectionnez la toile

La carte utilise le même principe, mais avec moins d'outils : B (crayon), M (sélection rectangulaire), S (tampon), V (déplacement) et Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Maintenez la touche Alt enfoncée pour zoomer sur la texture de la carte. La taille du pinceau sur la carte reste inchangée par rapport aux outils de peinture.

Chaque sprite possède huit bits de drapeau. Paint les place sur le sprite ; `fget` et `fset` les lisent dans le jeu.

PARAMÈTRES → ÉDITEUR DE PEINTURE : c'est ici que ce panneau est configuré :

- **Palette** — quelle banque est tirée, plus chargement et sauvegarde.
- **Aperçu** — RGB888 ou RGB565.
- **Curseur du pinceau** — le pointeur pendant que vous peignez.
- **Capture de pixels** — comment le pinceau se positionne sur les pixels.
- **Gros pinceau** — un gros pinceau se casse net en partant du coin supérieur gauche, ou de toute sa surface d'appui.
- **Pelure d'oignon** — un fantôme d'autres images, et la direction vers laquelle ce fantôme est tourné.
- **Zoom sur la carte** — pixels réels ou zoom libre.
- **Couleurs** — les couleurs de l'interface utilisateur de peinture.

CODE

Les onglets sont des fichiers distincts au sein du projet. Cliquez sur le nom d'un onglet pour le renommer ; appuyez sur Entrée pour confirmer. Plusieurs onglets, définissant chacun `_init`, `_update` ou `_draw`, sont assemblés dans l'ordre : le premier onglet s'exécute en premier, le dernier en dernier. Le code qui ne se trouve

pas dans un hook fait partie de `_init`. Une cartouche de jeu sans aucun hook est traitée comme un seul `_draw`.

Clé	Action
Flèches	Déplacez le curseur. La touche Maj étend la sélection.
Ctrl+Gauche / Droite	Par mot.
Accueil / Fin, Ctrl+Accueil / Fin	Début ou fin de ligne, ou de tabulation.
Languette	Insérez deux espaces. Dans la liste de saisie semi-automatique, acceptez le nom.
Entrer	Nouvelle ligne, indentée comme la précédente. Accepte une saisie semi-automatique.
Ctrl+C / X / V	Copier, couper, coller.
Ctrl+Z / Ctrl+Y	Annuler / rétablir.
Ctrl+A	Sélectionnez l'onglet.
Ctrl+D	Dupliquez la ligne ou la sélection vers le bas.
Ctrl+/	Commentez ou décommentez avec `--`.
Ctrl+F	Trouver. F3 permet de passer au coup suivant. Échap ferme la barre.
Ctrl+G	Allez à un numéro de ligne.
S'échapper	Ferme d'abord la fonction Rechercher, Aller à ou la liste des indices. Une seconde touche Échap permet de quitter l'éditeur et d'accéder au menu développeur.

Des suggestions s'affichent pendant que vous saisissez un nom de fonction. Appuyez sur Entrée ou Tabulation pour insérer le nom, et `(` si l'option COMPLET est activée dans les paramètres. PARAMÈTRES → ÉDITEUR DE CODE :

- **Rechercher** — Ctrl+F dans l'onglet actif, ou dans tous les onglets.
- **Hauteur du texte** — recadrer les glyphes à 5 px ou afficher le glyphe complet.
- **Taille des chaînes de caractères** — taille de la police des littéraux de chaînes de caractères.
- **Police des commentaires** — 3×5 ou la police Fusion de la langue d'interface utilisateur actuelle.
- **Complétez (** — indiquez si le type de complétion entre parenthèses.
- **Indications de code** — activées ou désactivées.
- **Langue des indications** — la langue du menu, ou l'anglais.
- **Espacement supplémentaire** — décalage supplémentaire sur les lignes vides.
- **Gouttière** — numéros de ligne.

- **Couleur** — arrière-plan, commentaires, tabulations, nombres, opérateurs, mots-clés, chaînes de caractères, fonctions du moteur, vos fonctions, variables, crochets, curseur, sélection, ligne d'erreur, avertissement.

SON

Le panneau comporte deux onglets. SFX correspond à un motif : une courte séquence de notes. SOUND correspond à un morceau : ces motifs sont répartis sur huit canaux. TIC indique le nombre de temps d'attente pour chaque pas. L'horloge a une fréquence de 120 tic par seconde, et un pas correspond à une double croche ; ainsi, 15 tic correspondent à 120 battements par minute. Un TIC plus élevé correspond à un motif plus lent. Deux motifs restent synchronisés lorsque l'un de leurs TIC est supérieur à l'autre : 15 et 30 sont synchronisés, 15 et 8 ne le sont pas.

Le piano joue lorsqu'une note est sélectionnée à la souris. Les touches blanches sont ZXCVBNM (CDEFGAB). Les touches noires sont SDGHJ (Do# Ré# Fa# Sol# La#). Survolez une valeur et saisissez un nombre, ou survolez-la et faites défiler la molette. Les hauteurs de notes, exprimées en code, vont de 0 (Do-0) à 63 (Ré#5).

Dans un motif, LOOP et LEN sont identiques. Si OUT est supérieur à 0, la boucle se déroule de IN à OUT, et le titre est LOOP. Si OUT est égal à 0 et IN est supérieur à 0, LEN est joué : ce nombre de notes est joué depuis le début, puis le motif se termine, et le titre est LEN. Si OUT et IN sont tous deux à 0, chaque note est jouée.

La quantité de sons du synthétiseur que vous pouvez modifier dépend de la profondeur de bits choisie pour la cartouche du jeu. Le menu PARAMÈTRES → ÉDITEUR D'EFFETS SONORES permet de régler cette profondeur, le passage automatique des notes (désactivé, enchaînées par notes ou par champs), le fonctionnement de la touche Espace (relancer la lecture ou activer/désactiver le démarrage/l'arrêt) et les couleurs de l'éditeur.

16 bits

Chaque étape possède une note, un volume, une onde et un effet. Le volume est compris entre 0 et 7. 0 correspond au silence.

Flots:

- **0 triangle**
- **1 scie inclinée**
- **2 scies**
- **3 carrés**
- **4 impulsions** — une impulsion étroite
- **5 organes**
- **6 bruits**
- **7 phaser**
- **8 PWM** — un carré dont la largeur se déplace

- **9 impulsions 25%**
- **10 sinus**
- **11 bruits métalliques**
- **12 FM cloche**
- **13 pincements**
- **14 superscie**
- **15 PWM (C64)**

Effets sur une seule note :

- **0 aucun**
- **1. Glissement** — la hauteur et le volume glissent vers la note suivante sur cette étape.
- **2 vibratos** — oscillations de hauteur
- **3e coup de pied** — le ton retombe dans le silence, le coup de pied habituel
- **4 fondu enchaîné**
- **5 fondu enchaîné**
- **Arpège de 6e degré** : cette note et les trois suivantes sont jouées, une toutes les 8 mesures. À la 8e mesure ou moins, toutes les 4 mesures.
- **7 arpèges rapides** — les mêmes quatre notes, toutes les 4 mesures, ou toutes les 2 mesures lorsque TIC est de 8 ou moins.

L'en-tête du modèle, au-dessus des notes :

- **TIC** — nombre de battements par minute, de 1 à 255. 15 correspond à 120 battements par minute. Saisissez le nombre ou incrémentez-le de 1.
- **BOUCLE ENTRÉE/SORTIE** — Si OUT est supérieur à 0, le motif se répète de IN à OUT. Si OUT est égal à 0, le titre devient LEN : IN indique le nombre de notes jouées, et 0 signifie le motif complet.
- **BRUIT** — le motif entier devient du bruit.
- **CRUSH** — une variante bourdonnante de chaque vague.
- **DTUNE** — 0 désactivé, 1 un refrain léger, 2 un refrain fort.
- **RVERB** — réverbération : 0 désactivée, 1 courte, 2 longue.
- **DAMPN** — un filtre passe-bas : 0 désactivé, 1 léger, 2 fort.
- **NIVEAU** — volume de l'ensemble du motif, de 0 à 7.
- **SWING** — pas chassés. Les pas impairs s'allongent un peu, les pas pairs un peu moins.
- **ÉTAPES** — nombre d'étapes utilisées par le motif : 32, 24, 16 ou 12.

8 bits

Chaque pas comprend une note et une onde. Les quatre ondes sont carrée, triangulaire, en dents de scie et bruit. L'en-tête comporte les options TIC, LOOP IN/OUT, FX et LEVEL. TIC peut prendre les valeurs suivantes : 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40 ou 60. FX correspond à un effet appliqué à l'ensemble du motif : 0 : aucun, 1 : vibrato, 2 : arpège, 3 : fondu. LEVEL est compris entre 0 et 3.

4 bits

Un pas correspond à une note. L'en-tête comporte TIC, LOOP (activé ou désactivé), WAVE (les quatre mêmes formes d'onde) et OCT (l'octave dans laquelle se trouvent les notes). TIC peut prendre les valeurs suivantes : 4, 6, 8, 10, 12, 15, 20 ou 30.

2 bits

Une étape peut être vide, un tambour, une frappe ou un écrasement. L'en-tête comporte les valeurs TIC et BASE. TIC peut être égal à 8, 15, 16 ou 30. BASE détermine la hauteur du tambour et de la frappe.

1 bit

Un pas correspond à une note. Le TIC est de 8, 15, 16 ou 30, soit les quatre mêmes vitesses que pour le 2 bits.

Chansons

L'onglet SON correspond au morceau en cours. Les huit colonnes représentent les canaux. Chaque cellule contient un identifiant de motif. La cellule 255, vide, indique que le canal correspondant est silencieux sur cette ligne. La colonne X contient une commande : continuer, arrêter, démarrer la boucle ou terminer la boucle. « Arrêter » met fin au morceau après la ligne. « Terminer la boucle » permet de revenir au dernier démarrage de boucle.

Une rangée se termine lorsque le motif le plus à gauche, dont la boucle est à 0, est terminé. Les motifs plus à droite n'allongent pas la rangée : un motif plus long est coupé, un motif plus court recommence à sa première étape, et ainsi de suite jusqu'à la fin de la rangée. Un motif dont la boucle est supérieure à 0 forme une boucle entre les points d'entrée et de sortie et ne termine pas la rangée. Si tous les motifs de la rangée forment une boucle, la rangée reste inchangée.

Raccourcis

Clé	Action
touches de piano	Prenez note de l'étape sélectionnée à l'aide de la souris.
Haut / Bas	Modifiez la valeur sous le curseur.
Espace	Lancez l'effet sonore ou le morceau ouvert de la première rangée. Démarrez, arrêtez ou redémarrez depuis les paramètres de l'éditeur d'effets sonores.
Ctrl+Z / Ctrl+Y	Annuler / rétablir.
Ctrl+C / Ctrl+X / Ctrl+V	Copier, couper, coller la sélection.

DONNÉES

Tables nommées. Chaque table comporte des lignes et des colonnes. Dans le code, `dget`, `dset`, `dstr`, `drows` et `dcols` utilisent ces tables. La numérotation des lignes et des colonnes commence à 0.

Types de colonnes, affichés en boucle dans l'en-tête :

- **booléen** — un bit.
- **2** — deux bits.
- **4** — quatre bits.
- **8** — un octet.
- **16** — 16 bits, pour les identifiants et les nombres plus grands.
- **spr** — un identifiant de sprite.
- **str** — texte, dessiné avec la police système.
- **str8** et **str10** — le même texte, dessiné plus haut.
- **f16** et **f32** — un nombre flottant demi-précision et un nombre flottant complet.

Lorsque le pointeur est positionné sur l'en-tête d'une colonne, Ctrl+molette permet de modifier le type de cette colonne. La molette permet de se déplacer horizontalement (colonnes) ou verticalement (lignes), selon l'option sélectionnée dans PARAMÈTRES → ÉDITEUR DE DONNÉES. Ctrl inverse ce sens de déplacement ; c'est pourquoi, lorsque la molette est réglée sur « Colonnes », Ctrl+molette sur un en-tête modifie le type de colonne.

Les touches fléchées permettent de se déplacer. La touche Entrée permet de modifier une cellule.
PARAMÈTRES → ÉDITEUR DE DONNÉES :

- **Molette** — faire défiler les colonnes ou les lignes.
- **Couleurs** — la grille, l'en-tête et la sélection.

Actifs

ASSETS importe et exporte l'élément ouvert, et non le projet entier. Les sprites utilisent le sprite ouvert dans PAINT, sur la feuille ouverte. La carte est la carte ouverte. Les effets sonores sont le motif ouvert. La musique est le morceau ouvert. Les données sont la table ouverte. Le code fait exception : chaque tabulation est écrite et relue.

Gentil	Déposer	Quels mouvements
Sprites	`.png`	Le sprite ouvert.
Carte	`.p37map`	La carte ouverte.
SFX	`.p37sfx`	Le motif ouvert.

Musique	`.p37mus`	Morceau ouvert. Les numéros de motifs restent inchangés. Exportez les effets sonores séparément si vous en avez besoin.
Données	`.p37dat`	La table ouverte.
Code	`.lua`	Chaque onglet.

Paramètres

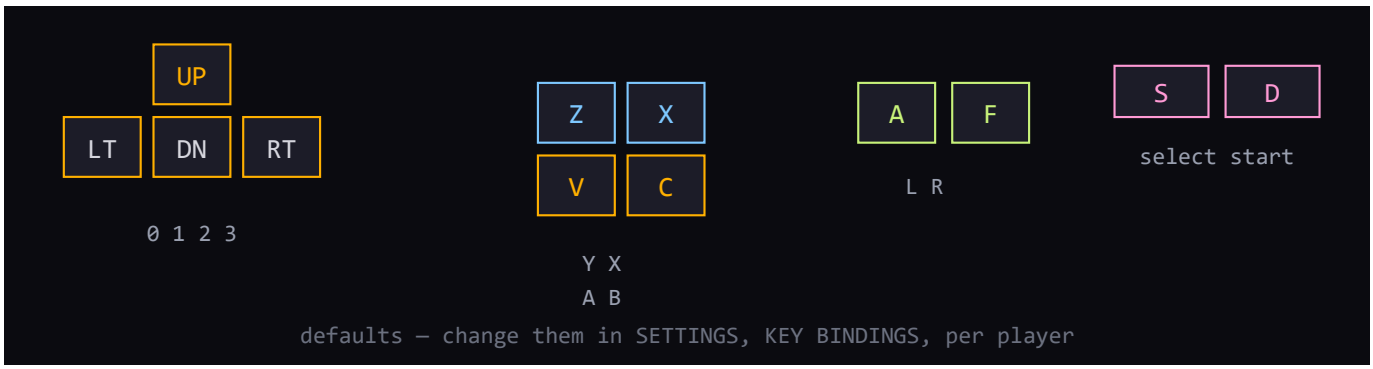
La liste racine :

Page	Ce qu'il stocke
PROFILS	Enregistrez et chargez tous les paramètres. Nouveau, renommer, écraser, supprimer. Copiez ou collez le profil sous forme de texte.
Menu	Fenêtre, touches, couleurs du menu, arrière-plan optionnel.
PANNEAU DE JEU	Aperçu et fenêtre de lecture.
ÉDITEUR DE CODE	Surlignage et astuces.
ÉDITEUR DE PEINTURE	Pinceau, palette, vue cartographique.
MONTEUR D'EFFETS SPÉCIAUX	Synthétiseur et saisie de notes.
ÉDITEUR DE DONNÉES	Sens de rotation et couleurs des roues.
LANGUE	Langue du menu et des indications.
LICENCES	Textes sous licence, y compris la police Fusion Pixel et SDL3.

MONDIAL:

- **MISE EN PAGE** — disposition des panneaux, ci-dessus.
- **VOLUME** — niveau principal, de 0 à 10.
- **RACCORDS DE TOUCHES** — touches pour chacun des 8 joueurs. Appuyez sur une touche pour la capturer. Directions : ABXYLR, sélectionner.
- **Présent** — ignorer les images inactives ou dessiner chaque image.
- **Plein écran** — sans bordure.
- **Actualiser** — FPS de la fenêtre. Pas à gauche et à droite : 60, 90, 120 et maximum. Appuyez sur Entrée pour saisir un nombre jusqu'à 1000. 0 signifie aucune limite.
- **Vsync** — attendre l'affichage entre les images.

- **Échelle** — multiplicateur de fenêtre entier.
- **Shaders** — CRT et filtres sur l'éditeur, pas sur le jeu.
- **Couleur** — texte du menu et surbrillance.
- **Arrière-plan** — une image PNG optionnelle, recadrée à 128 pixels, placée derrière le menu. L'option « Effacer » la supprime.
- **Curseur de l'éditeur** — forme du pointeur.
- **Souris** — pointeur flottant ou aligné sur les pixels.
- **Vitesse de l'interface utilisateur** — rapidité avec laquelle une action maintenue se répète dans une liste.
- **Explorateur** — la liste des fichiers commence à la racine de l'ordinateur, ou au dossier du fichier exe.



Clavier par défaut. A sert à confirmer dans les menus. B permet de revenir en arrière. En jeu, ces touches correspondent aux boutons 4 et 5. La touche 10 sert à sélectionner (S), et non à accéder aux menus.

Bouton	Rôle	Clé par défaut
0	Gauche	Flèche gauche
1	Droite	Flèche droite
2	En haut	Flèche vers le haut
3	Vers le bas	Flèche vers le bas
4	A, confirmer	V
5	B, dos	C
6	X	X
7	Y	Z
8	R	F
9	L	UN
10	Sélectionner	S

11	Commencer	D

Construire

Choisissez une plateforme : Windows, Linux, Mac, Android, iOS ou Web.

- **Nom** — nom du fichier de compilation.
- **Auteur** — sous Windows et Linux. Il s'agit du nom du dossier contenant les sauvegardes.
- **Identifiant de l'application Steam** — sous Windows et Linux. Un champ vide signifie que Steamworks n'est pas installé. Un identifiant renseigné associe `achieve(0)`…`achieve(15)` aux succès Steam ACH_0… ACH_15.
- **ID** — identifiant du bundle lorsque la plateforme en demande un.
- **Masquer le code** — sur les superpositions pour ordinateur et mobile. Cela ralentit le fonctionnement de la cartouche de jeu (`peek`). Il ne s'agit pas d'un verrou.
- **Plein écran** — le lecteur exporté démarre en plein écran.
- **Shaders** — jusqu'à trois couches intégrées au lecteur.
- **Télécharger le fichier .p37** — Web uniquement. Affiche une option de téléchargement dans le menu pause.
- **Icône** — une tuile extraite d'une feuille de sprites : feuille, taille, x, y. Ou une image PNG que vous recadrez.
- **Exporter** — sélectionnez le dossier. Le menu affiche « COMPILATION ENREGISTRÉE » ou « COMPILATION ÉCHOUÉE ».

Créez une version du jeu avec une connexion internet activée. L'écran d'accueil peut interroger le serveur, de manière anonyme, pour vérifier si le hachage de cette version est connu. Cette vérification est nécessaire pour deux raisons : elle me protège, en tant qu'auteur de PIX37, contre les copies de l'outil que je n'étais pas autorisé à distribuer gratuitement ; elle protège également le créateur du jeu. Le vol de jeux, notamment sur les plateformes mobiles comme le Play Store (Android) ou iOS, est une pratique de plus en plus courante. Le jeu se lance. Si l'appareil est hors ligne, l'écran d'accueil indique que le jeu n'est pas connecté. Ce message avertit également le joueur que la copie pourrait provenir d'une source illégale. Cette vérification, bien que simple, permet de signaler tout vol et vente de votre jeu sans votre autorisation sur des plateformes comme le Play Store (Android), Steam ou itch.io. Le serveur enregistre la date et l'heure du premier enregistrement du hachage, à côté de celui-ci, et ne les modifie pas par la suite. La date la plus ancienne correspond à la version originale. Un hachage enregistré après cette date correspond à une copie ultérieure.

Je souhaite que ce contrôle protège les créateurs de jeux. Le moteur du jeu leur facilite déjà grandement la tâche, et c'est l'un des aspects à prendre en compte. Je comprends que cela puisse susciter la controverse. Si la communauté décide qu'il est préférable d'abandonner ce micro-DRM, je suivrai son exemple.

Une ancienne cartouche de jeu provenant d'un éditeur précédent peut refuser de démarrer après un changement de langue. Ouvrez le projet `.pix37` dans cet éditeur et exportez-en un nouveau : `.p37`.

Connectez-vous à PIX37.PARTY

La connexion à PIX37.PARTY se fait directement dans l'application. Aucune page de connexion n'est disponible dans le navigateur. Vous vous connectez via un ticket de session Steam ou itch.io, à l'aide du pavé tactile et de la souris. Le service affiche un nom d'utilisateur unique de 2 à 16 caractères. La clé de licence et l'identifiant Steam ne sont pas visibles par les autres joueurs.

Après avoir accepté les conditions d'utilisation, vous accéderez aux sections suivantes : Favoris, Catalogue, Classement, Joueurs et votre profil.

La licence Développeur est celle de l'éditeur. Vous créez le jeu, vous le mettez en ligne et vous laissez des étoiles. La licence Joueur est celle du joueur. Les cœurs sont attribués par les joueurs qui participent à un jeu de course.

- **Catalogue** — Parcourez les jeux. Les filtres couvrent le genre (jusqu'à trois), la taille de l'écran, la taille de la cartouche de jeu et si le téléchargement inclut un projet `.pix37`.
- **Favoris** — jeux que vous avez marqués.
- **Téléversement** — nécessite une licence Développeur. Envoyez un fichier `.p37` et joignez éventuellement le projet `.pix37`. Un fichier `.p37` est un bytecode compressé. Difficile à lire, il peut être volontairement obscurci. Un fichier `.pix37` est le fichier de projet sur lequel vous travaillez : le code source, les images et le son, dans un format exploitable par d'autres personnes. Son envoi permet aux autres créateurs d'étudier le jeu plutôt qu'une cartouche compressée. Le titre, l'auteur, les genres, la jaquette, l'icône et les noms des succès sont inclus. La quantité de RAM transmise correspond à la mémoire maximale atteinte par la cartouche.
- **Commentaires et succès** — sur la page du jeu. Le déblocage d'un succès lorsque vous êtes connecté est enregistré sur votre compte ainsi que sur votre ordinateur.
- **Classement** des développeurs, triés par étoiles ou par cœurs. Chaque ligne affiche les deux scores. Cliquez sur un nom pour voir les jeux de cet auteur, classés par cœurs.
- **Joueurs** — comptes classés par succès débloqués.

Code

Le langage est un petit script de type Lua. Cette première partie commence par un projet vide, enseigne le langage, puis présente un court exemple pour chaque fonction.

Un premier programme

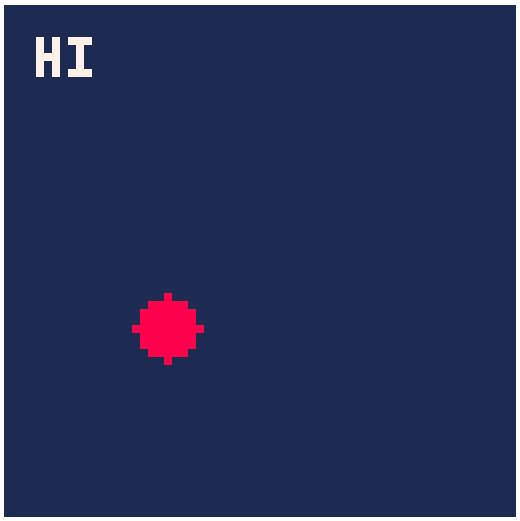
Créez un projet en 64×64, couleur 4 bits. Ouvrez CODE. Remplacez l'onglet par :

```
_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 lance le jeu. Les flèches directionnelles permettent de déplacer le cercle. La touche Échap arrête le jeu et retourne à l'éditeur.



Les mêmes trois appels : `cls(1)` dessine l'écran bleu marine, `print` dessine HI et `circfill` dessine le cercle rose.

Qu'est-ce qui se déroule, et quand ?

- `_init` s'exécute une fois.
- Le `_update` fonctionne à 60 Hz. Placez ici les commandes de mouvement et les boutons.
- `_draw` effectue des rendus. Sans argument de fréquence d'images, il effectue des rendus aussi rapides que l'ordinateur et le moniteur le permettent. `_draw(30)` limite le rendu à 30 images par seconde et laisse la logique à 60.

- `_set_fps(n)` modifie cette limite ultérieurement. 0 la supprime.

Les noms ne sont pas sensibles à la casse ; l'éditeur les stocke en minuscules. `function` et `fun` désignent le même mot-clé.

Langue

bases de la programmation

Un programme est une liste de lignes exécutées par la cartouche de jeu. Chaque en-tête indique d'abord la structure de la ligne, puis un court exemple. Une ligne commençant par `--` est un commentaire : une note à votre attention. Le jeu l'ignore.

Variables

Une variable est un nom qui contient une valeur. Vous choisissez ce nom. Les lignes de code suivantes peuvent la lire et y stocker une nouvelle valeur.

La forme est `name = value`. `local name = value` conserve ce nom à l'intérieur du bloc actuel, de sorte que le reste de la cartouche de jeu ne puisse pas le voir. Sans `local`, le nom est global et visible par tous les onglets.

```
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` signifie qu'il n'y a pas de valeur. `true` et `false` signifient respectivement « oui » et « non ». Un nom que vous n'avez jamais défini s'affiche sous la forme `nil`.

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

Tables

Une table regroupe plusieurs valeurs sous un même nom. Une liste utilise des nombres. Le premier élément est 1, et non 0. Un enregistrement utilise des noms. Une table peut contenir une autre table.

Forme de la liste : `{ value, value }`. Forme de l'enregistrement : `{ name = value, name = value }`. Une liste d'enregistrements est un tableau dont les éléments sont des tableaux.

```
-- 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
```

Condition physique et corps

Une condition est une question à laquelle on répond par oui ou par non, comme `hp > 0` ou `btnd(4)`. Le corps du bloc est l'ensemble des lignes exécutées lors de son entrée. `if` et toutes les boucles suivent la même structure : une condition, puis un corps, puis `end`. `repeat` se termine par `until` au lieu de `end`.

Si

`if` exécute son corps uniquement si la condition est vraie. `elseif` teste une autre condition. `else` s'exécute lorsque toutes les conditions sont fausses.

Forme : `if condition then`, puis le corps, puis `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
```

Boucles

Une boucle répète un bloc de code. `while` vérifie d'abord la condition et continue tant qu'elle est vraie. `repeat` exécute le bloc de code, puis vérifie à nouveau, de sorte que le bloc est toujours exécuté au moins une fois. `for` effectue un comptage. Le dernier chiffre est inclus. Sans troisième chiffre, le pas est de 1 ; par conséquent, si le début est déjà postérieur à la fin, le bloc n'est pas exécuté. Un décompte nécessite un pas négatif.

```
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
```

Marcher sur une table

`all` parcourt chaque élément d'une liste, en commençant par le premier. `pairs` parcourt chaque nom et la valeur stockée sous ce nom.

```
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
```

Fonctions

Une fonction est une recette nommée. Sa structure est `function name(arguments)`, suivie d'un corps, puis de `end`. On l'appelle en écrivant son nom entre parenthèses. `return` renvoie une valeur, et elle peut en renvoyer plusieurs. Les arguments sont des variables locales de cette fonction.

```
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` est une fonction qui peut accéder aux variables locales environnantes. Il s'agit d'une fermeture. Une table peut en contenir une, et on l'appelle avec un point.

```
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
```

Oui, non et choisir

`not` inverse une condition. `and` est vrai uniquement si les deux conditions sont vraies. `or` est vrai si au moins une des conditions est vraie. `a and b or c` sélectionne `b` si `a` est vrai, sinon `c`. `cond ? a : b` reprend le même principe en une seule ligne : `a` si la condition est vraie, `b` si elle est fausse.

```
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)
```

Mathématiques et texte

`+ - * /` représente l'addition, la soustraction, la multiplication et la division. `//` divise et conserve la partie entière. `%` affiche le reste. `^` représente la puissance. `..` combine deux textes. `#t` indique le nombre d'éléments dans une liste. Les comparaisons sont : `==` (identique), `~=` ou `!=` (différent), et `< > <= >=`. `x += 1` ajoute 1 à x. La même fonction s'applique à `-= *= /= ^= %=`.

```
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
```

Là où vit un habitant du coin

`local` est actif jusqu'à l'exécution de `end`, qui ferme son bloc. `if` / `end` constitue un bloc distinct. Vous pouvez spécifier une variable locale avant la ligne qui la définit, mais tant que cette ligne n'est pas exécutée, sa valeur est `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
```

Sauter

`::label::` marque une ligne. `goto label` y effectue un saut, mais uniquement à l'intérieur de la même fonction.

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

La mémoire, en bref

Les registres `peek` et `poke` permettent d'accéder aux pixels en temps réel via une adresse 16 bits. Ils ne conservent pas de copie de la cartouche de jeu. Les adresses `0x5F00` à `0x5FFF` correspondent aux registres suivants : caméra, clipping (lecture seule), couleur de dessin, motif de remplissage, les trois options de réaffectation de la palette et numéros de page. Les 65 280 octets situés de part et d'autre de cette zone délimitée constituent la fenêtre de pixels.

Lorsque l'écran tient dans ces 65 280 octets (64×64 et 128×128, mais pas 256×256), il démarre à l'adresse 0, ligne par ligne. L'espace restant correspond à la feuille de sprites sélectionnée par `0x5F41` (0 étant la feuille 0). Si cette feuille est plus grande que l'espace restant, `0x5F42` sélectionne la page et la carte n'est pas affichée. Si la feuille tient, la fin de la feuille correspond aux cellules de la carte sélectionnées par `0x5F43`, d'un ou deux octets chacune. `0x5F44` effectue un changement de page si la carte ne tient pas dans la fin de la feuille.

Lorsque l'écran est trop petit, la fenêtre entière occupe une page de 65 280 pixels. `poke(0x5F40, n)` permet de changer de page. `poke(0x5F41, n)`, avec n différent de 0, affiche la feuille n ; la feuille 0 est alors masquée. `poke(0x5F43, n)`, avec n différent de 0, affiche la carte n, qui est prioritaire sur la feuille. Dans ce mode, la carte 0 est uniquement accessible via `mget` et `mset`.

Un code `peek` qui ne s'applique ni à un pixel ni à un registre renvoie 0. Un code `poke` à cet emplacement est sans effet. Les indicateurs de sprite, les effets sonores et les tables de données ne sont pas accessibles dans cette fenêtre. Utilisez les codes `fget`, `sfxget` et `dget`.

utilisation pratique

Chaque fonction est répertoriée en bas de cette page, avec des exemples d'utilisation des fonctions PIX37. Les liens sous chaque exemple permettent d'obtenir plus d'informations sur les fonctions.



Mouvement fluide

Cartouche de jeu 128×128, vue de dessus. Le sprite 1 est déjà affiché. Les flèches permettent d'accélérer dans toutes les directions, et le frottement ralentit le joueur. Le sprite est à défilement : la moitié qui dépasse du bas est affichée en haut, et ainsi de suite pour les autres bords.

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



Vérification si le prochain coup est bloqué

Cartouche de jeu 64×64. Le sol et le mur de droite sont déjà présents sur la carte, représentés par le sprite 3, et le drapeau 0 est placé sur ce sprite. Les flèches permettent de déplacer le sprite 1. Le texte en haut indique « bloqué » lorsque le déplacement suivant chevaucherait une case pleine, et « libre » dans le cas contraire.

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



Une lingette avec rabat

Appuyez sur A. L'effet de balayage est dessiné par flip(), ligne noire par ligne, au lieu d'être exécuté par _draw(). _draw reste inactif pendant son exécution, de sorte que chaque ligne reste affichée à l'écran.

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



Tableau des fonctions (Classe)

Le sprite 1 est déjà dessiné. La fonction `make_monster` fonctionne comme une classe. Elle renvoie un tableau, et les méthodes `update` et `draw` conservent les coordonnées x et y du monstre. Le nombre en haut du tableau indique la distance parcourue.

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



Quatre façons de lire un bouton

A correspond au bouton 4. Maintenez-le enfoncé pour augmenter la charge. Relâchez-le pour que la charge revienne à 0. Les mouvements de haut en bas se répètent tant que le bouton est maintenu enfoncé, ce qui permet au menu de défiler. btnd correspond à la première image d'une pression, utilisée uniquement par cette cartouche de jeu pour indiquer le début du maintien du bouton.

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



Tirer avec la souris

Le sprite 1 est déjà dessiné au centre. Un point blanc fait office de curseur. Un clic gauche envoie un pixel vers ce point. La molette, située dans le coin, permet de régler la vitesse.

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



Saisissez un nom et confirmez-le

Appuyez sur A pour ouvrir le champ. Les flèches haut et bas permettent de modifier une lettre clignotante. A permet d'écrire, X d'effacer, Y d'insérer un espace et B de confirmer. Votre nom s'affiche ensuite.

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



Caméra sur le joueur

Cartouche de jeu 128×128. Le sprite 1 apparaît au centre d'un sol carrelé. Les flèches permettent le déplacement. La caméra maintient le sprite centré. Les axes x et y dans le coin ne défilent pas, car la fonction camera() sans argument ramène la vue à l'interface.

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



Quelques lignes d'un roman visuel

Les lignes de dialogue sont en anglais et en polonais. A permet de continuer ou de rester lorsqu'une option est affichée. B permet de quitter une option, et sinon, de changer de langue. X permet de changer de langue pendant qu'une option est affichée. La boîte de dialogue utilise une police de 8 pixels, ce qui permet d'afficher les caractères polonais.

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



Peindre avec la technique du peek and poke

Une cartouche de jeu 64×64, donc l'adresse 0 correspond au pixel en haut à gauche et la ligne suivante commence à 64. La cartouche de jeu écrit des pixels, copie une ligne, efface une ligne et imprime les valeurs qu'elle a lues.

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



Une liste que vous pouvez ajouter à

A ajoute un nombre, jusqu'à 8. B supprime le dernier. X supprime le premier. La liste s'affiche à l'écran.

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



Un nom décomposé

La cartouche de jeu divise ada,12,blue et imprime le nom, sa première lettre, 12, et le mot bleu.

`cls split sub chr ord tonum tostr print _draw`

 Jouez une note, puis changez-la.

Le motif 0 correspond à une seule note, et chaque répétition a la même intensité. A rejoue la note une fois terminée. B augmente la hauteur et joue cette note.

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

 Une chanson inscrite dans la cartouche du jeu

Deux motifs se répètent en boucle sur la ligne 0, un sur chacun des deux premiers canaux. La ligne reste à 0. ch0 et ch1 correspondent à la note de chaque canal. Modifier le motif, la longueur (LEN) et la boucle (LOOP) dans la cartouche de jeu permet de voir l'effet de ces commandes dans le panneau musical.

`songget songset music musicstat cls print _init _draw`

 Trois monstres dans les données

Le panneau DONNÉES comporte déjà une colonne Nom et une colonne PV, avec les éléments slime, bat et eye. Un A blesse le slime. Les nombres à droite représentent ses PV.

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

 N'oubliez pas le score

A ajoute un point et le sauvegarde. Il quitte le jeu et rouvre la cartouche : le nombre est toujours là. B l'oublie.

`load btnd save delsave cls print _init _update _draw`

  Un deuxième chapitre dans un autre fichier

Deux cartouches de jeu partagent la même sauvegarde. Partie 1 : A marque la fin du jeu. La partie 2, ouverte en premier, indique de terminer la partie 1. Ensuite, la partie 2 s'ouvre et B efface la sauvegarde pour pouvoir rejouer à la partie 1.

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

 Une réussite

Le sprite 8 est déjà affiché. Appuyez dix fois sur A. Le message indique le succès débloqué, et le sprite 8 reste affiché car il est débloqué. B supprime ce succès du fichier local et réinitialise le compteur.

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



Une pièce dans laquelle vous pouvez entrer

Une cartouche de jeu 256×256. Un point blanc sert de curseur. Le menu propose les options « Héberger », « Rejoindre » et la liste des serveurs. En tant qu'hôte, vous ouvrez une salle nommée « walk » et lancez la partie. En tant que joueur, vous faites défiler une adresse (chiffres, point, deux-points) et appuyez sur B pour vous connecter. La liste des serveurs se parcourt avec les flèches haut et bas, et vous pouvez rejoindre une ligne avec A. Dans le jeu, les flèches permettent de déplacer le sprite 1. Une pièce est partagée : l'hôte la déplace, et la ramasser ajoute 1 au score du joueur concerné et fait apparaître le suivant. Chaque sprite, y compris le vôtre, affiche son score. Dans le coin de l'écran, vous voyez le nombre de joueurs connectés, l'adresse LAN et l'adresse IP publique de l'hôte. Appuyez sur B pour quitter : vous et les autres joueurs retournez au menu, et le prochain hôte lance une nouvelle salle avec un score de 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`



Dialogue qui attend

La touche A affiche la ligne suivante. La fonction s'interrompt à chaque ligne jusqu'à ce que vous appuyiez de nouveau sur A.

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



Attendez une demi-seconde

Le mot passe de « … » à « salut » après environ une demi-seconde. L'affichage continue pendant la pause de la coroutine.

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



Des cercles qui arrivent un par un

Cinq cercles tombent du haut, espacés de deux cadres, puis retombent.

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



Deux images, tirées de l'horloge

Les sprites 1 et 2 représentent deux images du même personnage. L'image change quatre fois par seconde.

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

Chaque fonction

Les arguments entre crochets sont facultatifs. Les index des tables et des chaînes de caractères commencent à 1, sauf pour les lignes et les colonnes de données, qui commencent à 0. Les couleurs sont indexées par une palette.

Indice

_dessiner
_init
_mp
_mp_fail
_mp_join
_mp_leave
_mp_update
_set_fps
_mettre à jour
abdos
achclear
achdef
atteindre
atteint
ajouter
tout
affirmer
atan2
bande
mais pas
bor
btn
bdnd
btnp
btnu
bxor
caméra
plafond
chr
circ
remplissage circulaire
cliquez
cliquez
cliquez sur
agrafe
cls
cocréer
couleur
reprendre
parce que
costatus
compter
curseur
décolletés
del
charcuterie
sauvegarder
obtenir
drows
heure du coucher du soleil
dstr
fget
remplir
retourner
flr
pour chaque
fset
infos
paires i
coup de clé
clé
statistique clé
clé
ligne
charger
journal
lshr
carte
maximum
mémoire
ensemble de mémoires
récupérer
milieu
min
souris
mset
musique
musiquestat
annoncer
nappel
suivant
nhost
rejoindre
nkb
nkbstr
nlan
quitter
nliste
nécouter
remarque
njoueurs
envoyer
nstun
ord
ovale
remplissage ovale
emballer
paires
copain
palrgb
palt
appel téléphonique
coup d'oeil
coup d'oeil2
coup d'oeil4
performance
pget
piquer

poke2	sélectionner	chansonget
poke4	définir la carte	ensemble de chansons
imprimer	définir la palette	divisé
imprimer10	ensemblespr	spr
imprimer8	setex	carré
pset	tourner en rond	sable
rectifier	effets sonores	ensemble
rectifier	sfxget	SSPR
rnd	sfxhdr	statistique
rotl	ensemble d'effets	sous
rot	sfxstat	temps
correct	se procurer	ligne
rrectfill	signe	tonus
enregistrer	ch	tostr
emplacement de sauvegarde	statistiques d'exposition	déballer
écran	shr	whoami
faire défiler	péché	rendement

Cycle de vie

_init

`_init() ... end`

S'exécute une seule fois au démarrage de la cartouche de jeu. Le code situé en dehors de tout point d'entrée est également intégré à _init.

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

_update

`_update() ... end`

Logique du jeu. Appelée à 60 Hz, même si _draw est limité ou si le système hôte est plus lent. Consultez les boutons ici.

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

_draw

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

Dessine une image. Sans argument, ou avec 0, la fonction s'exécute à la fréquence d'affichage du système hôte. Indiquez n (de 1 à 999) pour limiter le nombre d'images par seconde. La fréquence de mise à jour reste fixée à 60 Hz.

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

_set_fps

`_set_fps(n)`

Modifie la limite de tirage (_draw cap) pendant l'exécution du jeu. 0 signifie illimité. n est limité à une valeur comprise entre 0 et 999.

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

flip

```
flip()
```

Affiche l'écran et attend l'image suivante. Utilisez cette fonction uniquement si vous créez votre propre boucle d'images. N'utilisez pas d'attente active après son exécution.

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

time

```
time()
```

Secondes depuis _init.

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

Saisir

btn

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

1 tant que le bouton b est maintenu, sinon 0. p est le joueur, 0..7. Sans arguments, il renvoie un masque de bits du joueur 0. Boutons : 0 gauche, 1 droite, 2 haut, 3 bas, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 sélectionner, 11 démarrer.

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

btnd

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

Vrai uniquement sur la trame _update lorsque le bouton b est enfoncé.

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

btnp

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

Valide à la pression, puis de nouveau en maintenant la touche enfoncée : délai de 30 ticks, puis toutes les 8 ticks. Sans bouton, renvoie un masque.

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

btnu

`btnu(b, [p])`

Vrai uniquement lorsque le bouton est relâché.

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

mouse

`mouse()`

Renvoie les coordonnées x et y de la souris en pixels du jeu.

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

clickl

`clickl([p])`

Bouton gauche de la souris. clickl() est maintenu. clickl(1) est le bord de pression.

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

clickr

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

Bouton droit de la souris. Même bord à 1 argument que clickl.

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

clickm

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

Bouton central ou clic sur la molette de défilement.

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

scroll

```
scroll()
```

Déplacement de la molette de la souris dans cette image. Le sens positif ou négatif dépend de la direction de la molette.

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

keyon

```
keyon()
```

Ouvre un champ de texte piloté par des boutons. Les flèches haut et bas permettent de sélectionner une lettre (A à Z). La touche A permet d'écrire la lettre. La touche X permet de supprimer. La touche Y insère un espace. La touche B permet de confirmer et de fermer le champ. Le clavier ne permet pas de saisir du texte dans ce champ.

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

keyoff

`keyoff()`

Masque le champ de texte.

```
keyoff()
```

keystr

`keystr([s])`

Saisie de texte dans le champ, puis confirmation par 1 dans le cadre B, puis changement de lettre (haut/bas). Cette lettre est vide une fois le champ fermé. keystr("") efface le texte.

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

keystat

`keystat()`

1 tant que le champ de texte est ouvert, sinon 0.

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

Dessin

screen

`screen()`

Résolution actuelle de l'écran de la cartouche de jeu en pixels. Renvoie la largeur, puis la hauteur. Il s'agit de la résolution du jeu, et non de celle de la fenêtre de l'éditeur.

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

cls

`cls([col])`

Efface l'écran. La couleur par défaut est 0.

```
cls(1)
```

color

`color(col)`

Définit la couleur de dessin utilisée lorsqu'un appel ultérieur omet la colonne. Sans argument, elle renvoie la couleur actuelle.

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

camera

`camera([x, y])`

Décale chaque appel de dessin. camera() se réinitialise à 0, 0.

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

clip

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

Limite le dessin à un rectangle. Un cinquième argument non nul intersecte la zone de découpage actuelle. La fonction `clip()` rétablit le plein écran. L'accès aux registres de découpage est en lecture seule.

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

cursor

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

Définit l'emplacement du prochain affichage sans coordonnées, et éventuellement la couleur. cursor() renvoie x, y.

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

pset

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

Définit un pixel d'écran. Compte pour la statistique px.

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

pget

`pget(x, y)`

Couleur du pixel de l'écran aux coordonnées x, y.

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

print

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

Affiche un texte de 3×5 caractères (UTF-8, extrait de la police Fusion). Sans les axes x et y, il s'affiche à l'emplacement du curseur et passe à la ligne suivante.

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

log

`log(...)`

Ajoute les arguments, séparés par des espaces, à un tampon de texte. pix37_run affiche ce tampon sur la sortie standard. Il ne dessine pas.

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

print8

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

Identique à l'impression, avec la police Fusion de 8 px.

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

print10

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

Identique à l'impression, avec la police Fusion de 10 px.

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

line

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

Ligne de x0,y0 à x1,y1. Deux arguments font suite à l'extrémité précédente. line() réinitialise cette chaîne.

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

circ

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

Contour du cercle. Rayon en pixels.

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

circfill

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

Cercle plein.

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

oval

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

Contour de l'ellipse à l'intérieur du cadre. Les coins sont inclus.

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

ovalfill

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

Ellipse pleine dans un cadre.

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

rect

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

Contour rectangulaire. Coins inclus.

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

rectfill

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

Rectangle rempli.

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

rrect

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

Contour d'un rectangle aux coins arrondis. x, y correspondent aux coordonnées du coin supérieur gauche. w et h représentent la taille en pixels ; elles doivent être supérieures à 0. r est le rayon des coins et vaut 0 par défaut. Il est limité à la moitié du côté le plus court.

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

rrectfill

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

Rectangle arrondi rempli. Mêmes arguments que rrect.

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

fillp

`fillp([p])`

Motif de remplissage 16 bits pour les formes remplies ultérieurement. 0 représente une couleur pleine. La fonction `fillp()` renvoie le motif actuel. Les bits parcourent le motif 4×4.

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

pal

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

Réaffecte la couleur c0 à c1. Le mode 0 correspond à la palette de dessin, le mode 1 à la palette d'affichage et le mode 2 à la palette de motifs de remplissage. pal() réinitialise.

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

palrgb

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

Définit une entrée de palette en RGB, chaque canal 0..255.

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

palt

`palt([col, t])`

Transparence. `palt(col)` ou `palt(col, true)` rend `col` transparent. `palt(col, false)` le rend opaque. `palt()` rend toutes les couleurs opaques.

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

setpalette

`setpalette(n)`

Sélectionne la banque de palettes n. setpalette() renvoie la banque actuelle.

```
setpalette(1)
```

tline

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

Trace une ligne sur l'écran tout en échantillonnant les pixels de la carte à partir de la position mx, my. mdx, mdy déplace l'échantillon de la carte le long de la ligne.

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

Sprites et cartes

spr

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

Dessine le sprite n aux coordonnées x, y. w et h représentent la taille en pixels (8 par défaut). fx et fy inversent horizontalement et verticalement lorsqu'ils sont différents de zéro. tex correspond à l'onglet de la feuille de sprites. rot représente l'angle de rotation en tours (0 à 1), dans le même sens que sin et cos. La valeur 0 conserve la rotation initiale du sprite.

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

sspr

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

Copie un rectangle de la feuille de sprites à l'écran, mis à l'échelle à dw par dh. Les valeurs par défaut conservent la taille source.

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

sget

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

Couleur d'un texel sur la feuille de sprites.

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

sset

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

Écrit un texel sur la feuille de sprites.

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

fget

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

Indicateurs de sprite. fget(n) renvoie l'octet d'indicateur. fget(n, f) est vrai lorsque le bit d'indicateur f (0..7) est défini.

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

fset

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

Définit les indicateurs du sprite. fset(n, v) écrit l'octet entier. fset(n, f, v) définit un bit.

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

setspr

```
setspr(n)
```

Sélectionne la feuille de sprites active. setspr() renvoie la feuille courante. Même banque que settex.

```
setspr(1)
```

settex

`settex(n)`

Alias de setspr.

```
settex(0)
```

map

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

Dessine la carte. Les cellules cx et cy sont dessinées à l'écran aux coordonnées sx et sy. cw et ch = 0 correspondent à la carte entière. Les calques sont acceptés ou ignorés. tex désigne la feuille de tuiles.

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

mget

`mget(cx, cy)`

Identifiant du sprite dans la cellule de carte cx, cy.

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

mset

`mset(cx, cy, n)`

Écrit le sprite n dans la cellule de carte cx, cy.

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

setmap

`setmap(n)`

Sélectionne la carte n. setmap() renvoie la carte actuelle. La carte 0 est une carte réelle.

```
setmap(1)
```

Mémoire

peek

`peek(addr)`

Lit un octet de la fenêtre de 64 Ko. La fenêtre correspond à l'écran actif, puis aux pixels du sprite, puis aux cellules de la carte lorsque l'écran tient dans 65 280 octets. Les écrans plus grands sont paginés avec `poke(0x5F40, page)`. Les adresses non mappées renvoient 0. Également écrit sous la forme `@addr`.

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

poke

`poke(addr, ...)`

Écrit des octets à l'adresse addr. Les arguments supplémentaires écrivent les octets suivants. Les adresses non mappées sont ignorées. Écrit également $addr et val. Toucher l'écran compte comme l'écriture d'un pixel.

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

peek2

`peek2(addr)`

Valeur non signée de 16 bits, petit-boutiste, à l'adresse.

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

poke2

`poke2(addr, val)`

Écrit une valeur non signée de 16 bits, petit-boutiste.

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

peek4

`peek4(addr)`

Valeur non signée de 32 bits, petit-boutiste.

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

poke4

`poke4(addr, val)`

Écrit une valeur non signée de 32 bits, petit-boutiste.

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

memcpy

`memcpy(dst, src, len)`

Copie len octets à l'intérieur de la même fenêtre de 64 Ko.

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

memset

`memset(dest, val, len)`

Remplit len octets à partir de dest avec val.

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

Mathématiques

abs

`abs(x)`

Valeur absolue.

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

flr

`flr(x)`

Arrondit vers moins l'infini.

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

ceil

`ceil(x)`

Arrondit vers l'infini positif.

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

max

`max(a, b)`

La plus grande des deux valeurs.

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

min

`min(a, b)`

La plus petite des deux valeurs.

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

mid

`mid(a, b, c)`

Valeur médiane des trois. Utile comme pince.

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

sgn

`sgn(x)`

Signe de x : -1, 0 ou 1.

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

sqrt

`sqrt(x)`

Racine carrée. Un argument négatif n'est pas un nombre.

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

sin

`sin(x)`

Le sinus de x est exprimé en tours : 0, 0,25, 0,5, 0,75 font un tour complet du cercle. Il ne s’agit pas de radians.

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

cos

```
cos(x)
```

Le cosinus, à son tour, suit la même convention que le péché.

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

atan2

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

Angle du vecteur dx, dy, en alternances de 0 à 1.

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

rnd

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

Nombre aléatoire dans [0, x). Par défaut, x vaut 1, donc rnd() renvoie 0..1. Pour un dé, utilisez flr(rnd(6))+1.

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

srand

```
srand(seed)
```

Initialise le générateur aléatoire afin qu'une séquence aléatoire ultérieure puisse être répétée.

```
srand(1)
```

band

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

Opération ET bit à bit. Également écrite a et b.

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

bor

`bor(a, b)`

OU bit à bit. Également écrit a | b.

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

bxor

`bxor(a, b)`

XOR bit à bit. Également écrit a ^^ b.

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

bnot

`bnot(a)`

NON bit à bit. Également écrit ~a.

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

shl

`shl(a, n)`

Décalage à gauche. S'écrit également a << n.

shr

`shr(a, n)`

Décalage arithmétique vers la droite. S'écrit également a >> n.

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

lshr

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

Décalage logique vers la droite. S'écrit également a >>> n.

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

rotl

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

Rotation des bits vers la gauche. Également écrit a <<> n.

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

rotr

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

Rotation des bits vers la droite. Également écrit >>< n.

Tableaux et valeurs

add

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

Ajoute v à la table t. Si i est donné, insère à cet index basé sur 1.

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

del

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

Supprime la première entrée égale à v.

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

deli

`deli(t, [i])`

Supprime l'élément d'index i et renvoie sa valeur. Sans i, supprime la dernière entrée.

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

all

`all(t)`

Itérateur parcourant les éléments du tableau, dans l'ordre. Les trous interrompent le parcours comme pour un tableau à base 1.

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

foreach

`foreach(t, fn)`

Appelle fn(v) pour chaque élément du tableau.

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

pairs

`pairs(t)`

Itérateur sur chaque clé et valeur, y compris les champs nommés.

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

ipairs

```
ipairs(t)
```

Itérateur sur la partie du tableau avec l'index basé sur 1.

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

next

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

Renvoie la clé et la valeur suivantes après k. next(t) commence au début. Utilisé par paires.

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

count

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

Longueur de la partie tableau. Si v est donné, compte le nombre d'éléments égaux à v.

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

pack

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

Regroupe les arguments dans un nouveau tableau.

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

unpack

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

Renvoie les valeurs t[i] à t[j] séparément. Les valeurs par défaut couvrent l'ensemble du tableau.

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

select

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

La fonction `select("#", ...)` renvoie le nombre d'arguments suivants. La fonction `select(i, ...)` renvoie les arguments à partir de l'argument `i`.

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

assert

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

S'arrête avec une erreur lorsque c est faux ou nul. msg remplace le texte par défaut.

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

pcall

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

La fonction f renvoie true et le résultat en cas de succès. En cas d'erreur, elle renvoie false et le texte d'erreur.

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

Cordes

chr

`chr(n)`

Un caractère d'un code d'octet 0..255.

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

ord

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

Codes d'octets des caractères s, de l'index i à j. Les index commencent à 1. Par défaut, le premier caractère est utilisé.

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

sub

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

Extraire la chaîne de caractères de i à j, indexée à partir de 1. Omettre j pour ne conserver que la fin.

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

split

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

Divise s en un tableau. sep utilise par défaut la virgule comme séparateur. Un nombre de séparateurs correspond à la largeur d'un segment. conv convertit les segments avec tonum lorsque cette option est activée.

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

tonum

`tonum(s, [base])`

Analyse un nombre. La base peut être comprise entre 2 et 36. Renvoie nil si le texte n'est pas un nombre.

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

tostr

`tostr(v, [digits])`

Convertit une valeur en texte. La fonction digits définit le nombre de décimales.

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

Audio

sfx

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

Joue le motif n et renvoie le canal. Omettez ch ou passez -1 pour sélectionner le premier canal libre. ch -2 arrête le motif n sur tous les canaux. n -1 arrête un canal, ou tous les canaux si ch est omis. n -2 permet à un motif en boucle de se terminer. sfx() sans argument renvoie -1.

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

sfxget

`sfxget(n, step)`

Renvoie un tableau d'une étape de motif : [1] hauteur, [2] forme d'onde, [3] volume, [4] effet. La hauteur est comprise entre 0 et 63 (de Do-0 à Ré#5). La forme d'onde est comprise entre 0 et 15. Le volume est compris entre 0 et 7, et 0 correspond au silence. L'effet est compris entre 0 et 7.

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

sfxset

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

Écrit une étape de motif. Les six arguments sont requis.

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

sfxhdr

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

En-tête du motif. `sfxhdr(n)` renvoie la table d'en-tête. `sfxhdr(n, key, val)` définit un champ : `speed`, `loop_in`, `loop_out`, `level`, `noise`, `crush`, `dtune`, `rverb`, `damkn`, `swing`, `steps`. Les nombres de 0 à 10 sélectionnent les mêmes champs dans cet ordre. La vitesse est exprimée en TIC (ticks par pas). Plus la valeur est élevée, plus le rythme est lent. 15 correspond à 120 battements par minute. Les synthétiseurs 16 bits acceptent les valeurs de 1 à 255. Les synthétiseurs plus étroits se positionnent automatiquement sur leur liste de TIC.

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

sfxstat

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

Identifiant du motif joué sur le canal ch, ou -1. Le canal par défaut est 0.

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

note

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

Joue une note unique sans motif. Hauteur : 0 à 63, durée : 0 à 15, volume : 0 à 7, effet : 0 à 7. La durée de la note est indiquée par « ticks », à la même fréquence de 120 Hz que le signal TIC. Sans « ticks », la note dure 15 secondes (une pas par défaut). L'effet s'applique pendant cette durée, puis le canal se coupe.

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

music

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

Lance le morceau n depuis le début. `music(-1)` arrête la lecture. Appeler `music` avec l'identifiant du morceau en cours de lecture le laisse en cours. `music()` renvoie l'identifiant du morceau en cours de lecture, ou -1. `fade` est exprimé en millisecondes : le morceau commence par un fondu, ou s'arrête progressivement lorsque n vaut -1. Les bits de masque activés correspondent aux canaux utilisés pour le morceau. 0 signifie que les huit canaux sont utilisés. Les autres canaux restent disponibles pour les effets sonores.

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

musicstat

`musicstat()`

Tableau du morceau en cours de lecture. `song` et `row` correspondent respectivement à l'identifiant du morceau et à la ligne, ou à -1 lorsqu'aucun son n'est joué. `step` et `pos` contiennent huit nombres. `step[1]` correspond au canal 0, `step[8]` au canal 7. `step` est l'indice de la note, ou à -1 lorsque le canal correspondant est silencieux. `pos` indique la position de la note, de 0 à 1. `tick` indique le nombre de ticks déjà joués pour cette ligne. `played` indique le nombre de lignes jouées depuis `music()`, y compris celle en cours de lecture.

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

songget

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

L'identifiant du motif est stocké dans une cellule de morceau. ch vaut 0 à 7. 255 indique que la cellule est vide. Il s'agit de l'appel en direct ; l'infobulle affiche encore un ordre d'arguments antérieur.

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

songset

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

Inscrit un identifiant de motif dans une cellule de chanson. Utilisez 255 pour effacer la cellule.

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

stat

`stat(n)`

Audio uniquement. 16 à 23 correspondent à l'identifiant des effets sonores sur le canal n-16. 24 à 31 correspondent au motif musical sur le canal n-24. Toute autre valeur de n renvoie 0. Le temps d'affichage et la RAM sont des informations de performance, et non des statistiques.

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

Tableaux de données

dget

`dget(t, row, col)`

Lit une cellule du tableau de données t du panneau DONNÉES. La ligne et la colonne commencent à 0.

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

dset

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

Écrit dans une cellule. Utilisez une chaîne de caractères pour une colonne de texte.

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

dstr

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

La cellule est au format texte. Les nombres sont convertis.

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

drows

```
drows(t)
```

Combien de lignes comporte le tableau t ?

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

dcols

```
dcols(t)
```

Combien de colonnes comporte le tableau t ?

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

Sauvegarde et succès

save

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

Enregistre la valeur v sous la clé k dans le fichier de sauvegarde. v peut être un nombre, une chaîne de caractères ou un tableau de ces éléments. Le fichier est compressé.

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

load

`load(k)`

Lit la clé k. Renvoie nil si la clé est absente.

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

delsave

`delsave(k)`

Supprime la touche k de la sauvegarde.

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

saveslot

`saveslot([name])`

Sélectionne le fichier de sauvegarde dans le dossier de l'auteur. Sans nom, le champ correspond au titre du jeu. Renvoie 1 si le fichier existe déjà.

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

achieve

`achieve(i)`

Débloque le succès i, de 0 à 15. Stocké localement. Une version Steam définit également ACH_i. Une session de catalogue le signale au compte du joueur.

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

achieved

```
achieved(i)
```

1 lorsque le succès i est déjà débloqué.

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

achdef

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

Message de félicitations pour le succès i, affiché pendant 15 secondes en bas de l'écran. spr est une tuile optionnelle de 8×8 pixels, tex sa feuille.

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

achclear

```
achclear(i)
```

Efface le succès i dans le fichier local. Un succès Steam ACH_i déjà défini reste actif.

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

Réseau

whoami

```
whoami()
```

Votre identifiant de joueur, compris entre 0 et 15, est utilisé pendant une session. Hors ligne, il est égal à -1.

```
id = whoami()
```

nhost

```
nhost()
```

1 si cette machine est l'hôte, sinon 0.

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

nplayers

```
nplayers()
```

Combien d'identifiants actifs existent dans 0..15 ?

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

nlisten

```
nlisten()
```

Démarre l'hébergement UDP et renvoie l'adresse LAN sous la forme ip:port.

```
addr = nlisten()
```

njoin

```
njoin(addr)
```

Se connecte via l'adresse IP ou l'adresse IP:port.

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

nleave

```
nleave()
```

Quitte la session. Les pairs voient _mp_leave avec la raison 0.

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

nsend

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

Envoie un message avec perte. Le nom correspond à l'événement. Jusqu'à quatre valeurs supplémentaires suivent. p (de 0 à 15) envoie un message privé à ce pair ; omettez p pour envoyer le message à tous.

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

ncall

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

Même fonctionnement que nsend, mais le message est renvoyé jusqu'à réception d'un accusé de réception. Trois échecs consécutifs entraînent la fermeture de la session.

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

nlan

```
nlan()
```

Adresse IP et port actuels du réseau local après l'écoute.

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

nstun

```
nstun()
```

Votre adresse IP publique et le port utilisés par STUN. Un invité situé sur un autre réseau peut tenter de s'y connecter.

```
pub = nstun()
```

setturn

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

TURN est utilisé en dernier recours lorsqu'un chemin direct échoue. La fonction setturn() sans arguments le désactive. L'adresse est au format hôte:port.

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

nlist

`nlist(game)`

Tableau des sessions ouvertes pour ce jeu. Chaque entrée comprend l'adresse, le nombre de joueurs et le nom.

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

nannounce

`nannounce(game)`

Un signal sonore indique que l'hôte apparaît dans la liste des joueurs. Programmez-le pendant que vous jouez.

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

nkb

`nkb([on])`

Champ d'adresse. nkb(1) l'ouvre, nkb(0) le masque, nkb() indique s'il est ouvert. Les touches haut et bas permettent de modifier un caractère, 0 à 9, un point ou deux-points. La touche A permet d'écrire. La touche X permet d'effacer. Le clavier ne permet pas de saisir de texte dans ce champ.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

Le texte saisi dans le champ adresse s'affiche, puis les caractères défilent. La fonction nkbstr("") efface le texte.

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

_mp

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

Vous écrivez cette fonction. Elle s'exécute à la réception d'un message. `from` correspond à l'identifiant de l'expéditeur. `name` correspond à l'événement. Jusqu'à quatre valeurs suivent.

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

_mp_join

`_mp_join(p)`

Vous écrivez cette fonction. Un pair a reçu l'identifiant p. Elle s'exécute également pour vous ; whoami() renvoie cet identifiant.

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

_mp_leave

`_mp_leave(p, why)`

Vous écrivez cette fonction. Le pair p est parti. Pourquoi 0 correspond à un départ normal ? Pourquoi 1 correspond à une perte de connexion ?

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

_mp_update

`_mp_update()`

Vous écrivez cette fonction. Elle s'exécute toutes les 15 images _update, soit environ 4 Hz. C'est un bon endroit pour envoyer les positions.

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

_mp_fail

`_mp_fail()`

Vous écrivez cette fonction. La session est interrompue : l'hôte a disparu ou ncall a échoué trois fois.

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

Coroutines

cocreate

`cocreate(f)`

Crée une coroutine à partir de la fonction f. Elle ne s'exécute pas tant que coresume.

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

coresume

`coresume(c, ...)`

Exécute la coroutine c jusqu'à ce qu'elle produise ou retourne une valeur. En cas de succès : renvoie vrai et les valeurs produites. En cas d'erreur : renvoie faux et l'erreur.

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

costatus

`costatus(c)`

"suspendu", "en marche" ou "mort".

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

yield

`yield(...)`

Redonne le contrôle à celui qui a appelé coresume. Les valeurs deviennent les résultats de cet appel à coresume.

```
yield(1)
```

Mesure

info

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

Affiche l'interface utilisateur en temps réel dans _draw : fps (moyenne sur 1 seconde), upd, drw, px, max, ram, peak, ramc et net lorsqu'une session est active. info() utilise le curseur d'impression. info(col) définit uniquement la couleur. info(x, y) ou info(x, y, col) la positionne. Les nombres utilisent B, k ou m pour s'adapter à un petit écran. L'appel lui-même n'est pas comptabilisé en pixels.

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

showstats

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

Même interface que les informations, dessinée automatiquement à 0, 0 après _draw, ignorant la caméra et le clipping. showstats(0) la masque. showstats(1) ou showstats() l'affiche.

```
showstats(1)
```

perf

```
perf(n)
```

Un seul chiffre pour l'interface, à personnaliser. 0 fps, 1 upd, 2 drw, 3 px, 4 max (image la plus lourde de cette seconde), 5 ram, 6 ramc (RAM plus la cartouche de jeu), 7 pic de RAM depuis le début du jeu.

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