PIX37 사용 설명서

편집 후 코드를 작성하세요. 책의 오프라인 사본은 pix37.party/docs에서 확인할 수 있습니다.

편집자

PIX37 DEV는 에디터입니다. 여기서 게임을 그리고, 쓰고, 사운드를 만들고, 게임 카트리지나 독립 실행형 버전을 내보낼 수 있습니다. 또한 게임 카트리지를 재생할 수 있으며, PIX37.PARTY를 열어 게임 카트리지를 다운로드하고 카탈로그를 이용할 수 있습니다. PIX37 DEV EDUCATIONAL은 동일한 에디터이며 무료입니다. 게임 빌드 기능, 게임 카트리지 내보내기 기능, PIX37.PARTY 접근 권한은 없습니다. PIX37 Console은 플레이어입니다. `.p37` 게임 카트리지를 재생하며, PIX37.PARTY를 열어 게임 카트리지를 다운로드하고 카탈로그를 탐색할 수 있습니다.

프로젝트 및 게임 카트리지

`.pix37` 파일은 프로젝트 파일입니다. 소스 코드 탭, 스프라이트 시트, 맵, 사운드 및 데이터 테이블 등이 포함됩니다. `.p37` 파일은 플레이어가 실행하는 게임 카트리지 파일입니다. 이는 압축된 바이트코드로 구성됩니다.

저장 파일과 업적 파일은 앱 데이터 폴더 옆에 있는 제작자 폴더에 저장됩니다. Windows에서는 저장 파일의 경우 `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` 경로에, 업적 파일의 경우 `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` 경로에 있습니다. Linux에서는 동일한 파일이 `~/.local/share/PIX37/Mira/Pong.p37s` 및 `~/.local/share/PIX37/Mira/Pong.ach` 경로에 있습니다. Ada는 컴퓨터 계정 이름이고, Mira는 제작자 이름이며, 해당 폴더 이름이 Mira입니다. Pong은 저장 슬롯 이름으로, 기본값은 게임 제목입니다. 저장 파일은 `.p37s` 파일에 저장되고, 첫 번째 업적 달성 파일은 `.ach` 파일에 `save` 됩니다. 파일이 이미 존재하는 경우, 게임 카트리지가 시작될 때 실행 프로그램이 해당 파일을 읽어옵니다.

개발자 메뉴

Esc 키를 누르면 개발자 메뉴가 열립니다. 게임 실행 중에 Esc 키를 누르면 에디터로 돌아갑니다. Enter 키 또는 A 버튼(기본값은 V)으로 확인합니다. 뒤로 가기는 B 버튼(기본값은 C)입니다.

프로젝트가 로드되지 않은 경우 메뉴는 다음과 같습니다: 프로젝트 생성, 프로젝트 불러오기, 빌드, 게임 카트리지 불러오기, PIX37.PARTY에 연결, 설정.

프로젝트가 로드된 상태에서의 순서는 다음과 같습니다: 프로젝트 저장, 프로젝트 불러오기, 프로젝트 생성, 에셋, 빌드, 게임 카트리지 불러오기, PIX37.PARTY에 연결, 설정.

Ctrl+S를 누르면 열려 있는 프로젝트가 저장되고, Ctrl+R을 누르면 실행됩니다.

프로젝트를 생성하세요

프로젝트 생성 버튼을 누르면 프로젝트 제작을 시작하기 전에 게임 카트리지를 요청합니다.

- **비디오** — 1비트(2색), 2비트(4), 4비트(16), 8비트(256).
- **오디오** — 신디사이저의 샘플 깊이: 16, 8, 4, 2 또는 1비트.
- **화면** 크기 설정에는 64×64, 128×128, 160×90, 256×256, 320×180이 포함되어 있습니다. 사용자 지정 크기를 선택하면 1×1부터 512×512까지 원하는 크기를 지정할 수 있습니다.
- **팔레트** — 색상 모음입니다. 첫 번째 팔레트는 아래와 같은 16색 세트입니다. '팔레트 불러오기'를 선택하면 내장 팔레트 세트가 적용됩니다. 여기에는 PICO-8 16, PICO-8 32, Game Boy, 1비트, CGA, Sweetie-16, 그리고 Lospec에서 가져온 더 긴 목록(Resurrect 64, Apollo, Endesga 32 및 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century 등)이 포함됩니다. 팔레트 파일을 불러오거나, 직접 만든 PNG 파일을 불러와 이미지에서 팔레트를 추출할 수도 있습니다.



`cls` , `pset` , `spr` 등에서 사용되는 색상 인덱스입니다. 팔레트 편집기에서 모든 인덱스를 다시 칠할 수 있습니다.

다섯 개의 패널

에디터는 게임, 그림판, 코드, 사운드, 데이터 순서로 구성되어 있습니다. 맵은 그림판 안에 있고, 효과음과 음악은 사운드판 안에 있습니다. 각 패널을 클릭하면 해당 패널에서 작업할 수 있습니다. 코드 패널에 포커스가 있을 때는 사운드와 데이터 패널이 키 입력을 무시하므로, 키보드 입력으로 피아노가 연주되지 않습니다.



전체 레이아웃은 `1:1:4:1:1` 로 작성되었습니다. 각 숫자는 해당 패널이 행에서 차지하는 크기를 나타냅니다: 게임, 페인트, 코드, 사운드, 데이터. 작은 화면은 128x128 픽셀 셀의 중앙에 배치됩니다. 128픽셀보다 큰 화면은 게임 열의 너비를 넓힙니다.

설정 → 전체 → 레이아웃에서 레이아웃을 순환적으로 변경할 수 있습니다.

암호	당신이 보는 것
`1:1:4:1:1`	모든 것. 게임과 사운드는 겹쳐져 있고, 페인트와 데이터는 겹쳐져 있으며, 코드는 오른쪽에 있습니다.
`1:1:4:1:1 2X`	동일한 배열입니다. CODE는 두 배의 밀도로 그려지므로 두 배의 텍스트가 들어갑니다.
`1:0:1:0:1`	게임, 데이터, 코드.
`1:0:1:0:1 2X`	더 밀도가 높은 코드를 사용한 동일한 구성입니다.
`0:1:1:1:1`	게임 미리보기 없음. 페인트, 코드, 사운드, 데이터.
`0:1:1:1:1 2X`	더 밀도가 높은 코드를 사용한 동일한 구성입니다.
`0:1:4:1:2`	스튜디오. 넓은 코드, 더 긴 데이터 스트립, 페인트 및 사운드.
`0:1:4:1:2 2X`	더 밀도가 높은 코드를 사용한 동일한 구성입니다.

`0:0:4:0:4`	코드와 데이터만 해당되며, 둘 다 용량이 큽니다.
`0:0:4:0:4 2X`	더 밀도가 높은 코드를 사용한 동일한 구성입니다.
`0:1:4:1:0`	페인트, 사운드, 그리고 광범위한 코드.
`0:1:4:1:0 2X`	더 밀도가 높은 코드를 사용한 동일한 구성입니다.
`0:4:0:0:0`	페인트만 사용, 256×256, 용지 한 픽셀에 캔버스 한 픽셀을 대응시킵니다.

`2X` 행은 동일한 패널을 유지합니다. 코드 패널은 두 배의 해상도로 그려진 후 동일한 셀에 맞게 축소되므로 텍스트, 탭 및 여백은 높이가 절반으로 줄어들고 화면에 표시되는 소스 코드는 두 배로 늘어납니다.

게임

실행 중인 게임 카트리지의 미리보기 화면입니다. 게임 패널을 클릭하거나 Ctrl+R 키를 눌러 미리보기를 실행하세요. 설정 → 게임 패널에서 패널을 구성할 수 있습니다.

- **미리보기** — 대기 색상, 연속 표시 또는 편집 시 자동 새로 고침.
- **새 창에서 실행** — 게임이 편집기 옆에 열립니다.
- **디스플레이** — RGB888 또는 RGB565로 미리보기를 제공합니다.
- **게임 64** — 64×64 화면은 셀 중앙에 1:1 비율로 배치되거나, 셀을 채우도록 2배로 확대됩니다.
- **셰이더** — 게임 내 CRT 및 필터는 에디터의 필터와는 별개입니다.
- **색상** — 패널의 배경색입니다.

페인트

스프라이트 시트와 같은 패널의 맵 부분에는 타일 맵이 표시됩니다. 시트는 화면보다 클 수 있습니다. 스프라이트 미리보기 크기는 8, 16, 32, 64, 128픽셀입니다(시트 크기가 128픽셀 이상일 경우에만 128픽셀). 마우스 휠을 스프라이트 미리보기 위로 가져가면 확대/축소할 수 있습니다. 브러시 크기는 1, 2, 3, 4픽셀입니다.

열쇠	도구
비	연필
중	직사각형 선택
큐	올가미
에스	우표
G	버킷

나	점안액
다섯	이동하다
L / Shift+L	직선/곡선
U / Shift+U	속이 빈 직사각형 / 속이 빈 타원
F / Shift+F	채워진 직사각형 / 채워진 타원
디	떨림
[ ]	작은/큰 브러시
Ctrl+Z / Ctrl+Y	실행 취소/다시 실행. Ctrl+Shift+Z도 다시 실행 기능을 합니다.
Ctrl+C / Ctrl+X / Ctrl+V	복사, 잘라내기, 붙여넣기
Ctrl+A	캔버스를 선택하세요

지도는 더 적은 도구를 사용하지만 동일한 아이디어를 활용합니다. B는 연필, M은 사각형 선택, S는 도장, V는 이동, Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V는 이동 단축키입니다. Alt 키를 누르면 지도 텍스처를 확대/축소할 수 있습니다. 지도에서 브러시 크기는 페인트 도구에 따라 조정됩니다.

각 스프라이트는 8개의 플래그 비트를 가지고 있습니다. Paint 함수는 스프라이트에 이 플래그 비트를 설정하고, 게임 내에서 `fget` 및 `fset` 사용하여 이 플래그 비트를 읽습니다.

설정 → 페인트 편집기에서 해당 패널을 구성할 수 있습니다.

- **팔레트** — 어떤 은행에서 인출할지, 그리고 입금 및 저축 여부를 표시합니다.
- **미리보기** — RGB888 또는 RGB565.
- **브러시 커서** - 그림을 그릴 때 나타나는 포인터입니다.
- **픽셀 스냅** — 브러시가 픽셀에 맞춰지는 방식입니다.
- **큰 붓** — 큰 붓은 왼쪽 윗부분이나 전체 발자국에서 부러집니다.
- **양파 껍질** — 다른 프레임의 잔상, 그리고 그 잔상이 향하는 방향.
- **지도 확대/축소** - 실제 픽셀 단위 확대/축소 또는 자유 확대/축소.
- **색상** — 페인트 UI 색상입니다.

암호

탭은 프로젝트 내의 개별 파일입니다. 탭 이름을 클릭하여 이름을 변경할 수 있으며, Enter 키를 누르면 변경 사항이 확정됩니다. `_init` , `_update` 또는 `_draw` 정의하는 여러 탭은 탭 순서대로 연결됩니다. 첫 번째 탭이 먼저 실행되고 마지막 탭이 나중에 실행됩니다. 훅 내부에 있지 않은 코드는 `_init` 의 일부가 됩니다. 훅이 전혀 없는 게임 카트리지는 하나의 `_draw` 로 처리됩니다.

열쇠	행동
화살표	커서를 이동하세요. Shift 키를 누르면 선택 영역이 확장됩니다.
Ctrl+왼쪽/오른쪽	단어로.
홈/엔터, Ctrl+홈/엔터	줄의 시작 또는 끝, 또는 탭의 시작 또는 끝.
꼬리표	두 칸을 띄어쓰기하세요. 자동 완성 목록에서 이름을 수락하세요.
입력하다	새 줄, 이전 줄처럼 들여쓰기. 자동 완성 기능을 지원합니다.
Ctrl+C / X / V	복사, 잘라내기, 붙여넣기.
Ctrl+Z / Ctrl+Y	실행 취소/다시 실행.
Ctrl+A	탭을 선택하세요.
Ctrl+D	해당 줄 또는 선택 영역을 아래쪽으로 복제합니다.
Ctrl+/	-- 사용하여 댓글을 달거나 댓글을 취소하세요.
Ctrl+F	찾기. F3 키를 누르면 다음 검색 결과로 이동합니다. Escape 키를 누르면 진행 표시줄이 닫힙니다.
Ctrl+G	줄 번호로 이동하세요.
탈출하다	먼저 찾기, 이동 또는 힌트 목록을 닫습니다. 두 번째 Esc 키를 누르면 편집기를 종료하고 개발자 메뉴로 이동합니다.

함수 이름을 입력하는 동안 힌트가 나타납니다. Enter 또는 Tab 키를 누르면 함수 이름이 입력되고, 설정에서 "완료( )가 켜져 있으면 (기호가 표시됩니다. 설정 → 코드 편집기"를 참조하세요.

- **검색** — 현재 탭에서 Ctrl+F를 누르거나, 모든 탭에서 Ctrl+F를 눌러 검색할 수 있습니다.
- **텍스트 높이** — 글리프를 5픽셀로 자르거나 전체 글리프를 표시합니다.
- **문자열 크기** — 문자열 리터럴의 글꼴 크기입니다.
- **주석 글꼴** — 3×5 또는 현재 UI 언어의 Fusion 글꼴.
- **완료(** — 괄호를 자동 완성할지 여부.
- **코드 힌트** - 켜기 또는 끄기.
- **힌트 언어** — 메뉴 언어 또는 영어.
- **캐럿 간격** - 빈 줄에 추가 오프셋이 적용됩니다.
- **홈** — 줄 번호.
- **색상** — 배경, 주석, 탭, 숫자, 연산자, 키워드, 문자열, 엔진 함수, 사용자 정의 함수, 변수, 괄호, 캐럿, 선택 영역, 오류 줄, 경고.

소리

패널에는 두 개의 탭이 있습니다. SFX는 하나의 패턴, 즉 짧은 음표들의 연속입니다. SOUND는 곡으로, 이러한 패턴들을 8개의 채널에 배치합니다. TIC는 한 단계가 기다리는 시간(틱)입니다. 시계는 초당 120틱이고, 한 단계는 16분음표이므로 15틱은 분당 120비트입니다. TIC 값이 높을수록 패턴은 느려집니다. 두 패턴은 한 TIC가 다른 TIC를 나누더라도 박자가 유지됩니다. 15와 30은 박자가 유지되지만, 15와 8은 박자가 유지되지 않습니다.

마우스로 음표를 선택하면 피아노가 연주됩니다. 흰 건반은 ZXCVBNM(CDEFGAB)이고, 검은 건반은 SDGHJ(C# D# F# G# A#)입니다. 아무 값 위에 마우스를 올려놓고 숫자를 입력하거나, 마우스 휠을 돌려 음높이를 조절할 수 있습니다. 코드에서 음높이는 0(C-0)부터 63(D#5)까지입니다.

한 패턴에서 LOOP와 LEN은 동일한 두 숫자입니다. 0보다 큰 OUT은 IN부터 OUT까지 반복되며 제목은 LOOP입니다. OUT이 0이고 IN이 0보다 크면 LEN이 됩니다. 즉, 해당 음표 수만큼 처음부터 연주한 후 패턴이 끝나고 제목은 LEN입니다. 둘 다 0이면 매 단계마다 연주됩니다.

신디사이저 편집 가능 범위는 게임 카트리지에 선택된 비트 심도에 따라 달라집니다. 설정 → 효과음 편집기에서 비트 심도, 음표 자동 진행 여부(끔, 음표 체인, 필드 체인), Space 키가 재생을 다시 시작하거나 시작/정지를 전환하는지 여부, 편집기 색상을 설정할 수 있습니다.

16비트

각 단계에는 음표, 볼륨, 파형 및 효과가 있습니다. 볼륨은 0에서 7까지이며, 0은 무음입니다.

파도:

- **0 삼각형**
- **1 기울어진 톱**
- **2개의 톱**
- **3제곱**
- **4펄스** — 좁은 펄스
- **5개 기관**
- **6 노이즈**
- **7 페이저**
- **8 PWM** — 폭이 변하는 정사각형
- **9펄스 25%**
- **10 사인**
- **11 금속성 소음**
- **12 FM 벨**
- **13 뽑기**
- **14 슈퍼쏘**
- **15 PWM (C64)**

단일 음에 미치는 영향:

- **0 없음**
- **1 슬라이드** — 이 단계를 건너뛰면 음높이와 음량이 다음 음으로 부드럽게 이동합니다.
- **2 비브라토** — 음정 흔들림
- **3 드롭** - 피치가 침묵으로 떨어지고, 평소의 킥 소리가 남습니다.
- **4 페이드 인**
- **5단계로 서서히 사라짐**
- **6 아르페지오** — 이 음과 다음 세 음을 8틱마다 하나씩 연주합니다. 8틱 이하에서는 4틱마다 연주합니다.
- **7번 빠른 아르페지오** - 동일한 네 음을 4틱마다, 또는 TIC가 8 이하일 때는 2틱마다 연주합니다.

패턴 헤더는 메모 위에 있습니다.

- **틱(TIC)** — 단계당 틱 수, 1부터 255까지. 15는 분당 120회 박동을 의미합니다. 숫자를 직접 입력하거나 1씩 단계적으로 설정하세요.
- **LOOP IN / OUT** — OUT 값이 0보다 크면 패턴이 IN에서 OUT으로 반복됩니다. OUT 값을 0으로 두면 제목이 LEN이 됩니다. IN은 연주되는 음표의 개수를 나타내고, 0은 전체 패턴을 의미합니다.
- **노이즈** — 전체 패턴이 노이즈가 됩니다.
- **CRUSH** — 각 웨이브의 활기 넘치는 변형.
- **DTUNE** — 0은 꺼짐, 1은 약한 코러스, 2는 강한 코러스.
- **RVERB** — 리버브: 0 끔, 1 짧음, 2 김.
- **DAMPN** — 저역 통과 필터: 0은 꺼짐, 1은 약함, 2는 강함.
- **레벨** — 전체 패턴의 볼륨, 0~7.
- **스윙** — 발을 질질 끌며. 홀수 발은 조금 더 오래, 짝수 발은 조금 더 빨리 멈춥니다.
- **단계 수** — 패턴에 사용되는 단계 수: 32, 24, 16 또는 12.

8비트

각 단계는 음표와 파형으로 구성됩니다. 파형은 사각파, 삼각파, 톱니파, 노이즈의 네 가지입니다. 헤더에는 TIC, LOOP IN/OUT, FX, LEVEL이 있습니다. TIC는 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60 중 하나입니다. FX는 전체 패턴에 적용되는 하나의 효과입니다: 0은 효과 없음, 1은 비브라토, 2는 아르페지오, 3은 페이드 아웃입니다. LEVEL은 0에서 3까지의 값입니다.

4비트

스텝은 음표 하나를 의미합니다. 헤더에는 TIC, LOOP(켜기/끄기), WAVE(동일한 네 가지 파형), OCT(음표가 속한 옥타브)가 있습니다. TIC는 4, 6, 8, 10, 12, 15, 20, 30 중 하나입니다.

2비트

스텝은 비어있는 상태, 드럼, 히트 또는 크러시일 수 있습니다. 헤더에는 TIC와 BASE가 있습니다. TIC는 8, 15, 16 또는 30입니다. BASE는 드럼과 히트의 높이를 설정합니다.

1비트

스텝은 음표입니다. TIC는 8, 15, 16 또는 30으로, 2비트와 동일한 네 가지 속도입니다.

노래

'사운드' 탭은 현재 재생 중인 곡입니다. 8개의 열은 채널을 나타냅니다. 각 셀은 패턴 ID이며, 255(빈 셀)는 해당 행에서 해당 채널이 무음임을 의미합니다. 'X' 열은 '계속', '정지', '반복 시작', '반복 종료' 명령어를 나타냅니다. '정지'는 해당 행 이후의 곡 재생을 종료하고, '반복 종료'는 마지막 반복 시작 지점으로 돌아갑니다.

LOOP OUT이 0인 가장 왼쪽 패턴이 끝나면 해당 행이 종료됩니다. 오른쪽 패턴들은 해당 행을 더 길게 만들지 않습니다. 더 긴 패턴은 잘리고, 더 짧은 패턴은 첫 번째 단계부터 다시 시작하여 행이 끝날 때까지 계속됩니다. LOOP OUT이 0보다 위에 있는 패턴은 LOOP IN과 LOOP OUT 사이에 루프를 형성하며 행을 끝내지 않습니다. 행의 모든 패턴이 루프를 형성하는 경우, 해당 행은 그대로 유지됩니다.

바로가기

열쇠	행동
피아노 건반	마우스로 선택한 단계에 메모를 작성하세요.
위/아래	커서 아래 값을 변경하세요.
공간	열려 있는 효과음 또는 첫 번째 줄에 있는 열려 있는 노래를 재생합니다. 효과음 편집기 설정에서 시작/정지 또는 다시 시작을 할 수 있습니다.
Ctrl+Z / Ctrl+Y	실행 취소/다시 실행.
Ctrl+C / Ctrl+X / Ctrl+V	선택한 부분을 복사, 잘라내기, 붙여넣기 하세요.

데이터

이름이 지정된 테이블들입니다. 각 테이블은 행과 열로 구성됩니다. 코드에서 `dget` , `dset` , `dstr` , `drows` , `dcols` 이러한 테이블을 사용합니다. 코드에서 행과 열 번호는 0부터 시작합니다.

헤더에서 순환되는 열 유형:

- **bool** — 1비트.
- **2** — 2비트.
- **4** — 4비트.
- **8** — 바이트.
- **16** — 16비트, ID 및 더 큰 숫자에 사용됩니다.
- **spr** — 스프라이트 ID입니다.
- **str** — 시스템 글꼴로 그려진 텍스트입니다.
- **str8** 과 **str10** 은 같은 텍스트를 더 크게 그린 것입니다.

- **f16** 과 **f32** 는 각각 반정밀 부동소수점과 완전 부동소수점입니다.

열 머리글에 포인터를 놓고 Ctrl 키를 누른 상태에서 스크롤하면 해당 열의 유형이 변경됩니다. 일반 스크롤은 설정 → 데이터 편집기에서 선택한 방향에 따라 열 또는 행을 따라 이동합니다. Ctrl 키는 이 방향을 반대로 바꾸므로, 마우스 휠이 열로 설정된 상태에서 머리글에 마우스 포인터를 놓고 Ctrl 키를 누른 상태에서 스크롤하면 유형이 변경됩니다.

화살표 키로 이동합니다. Enter 키를 누르면 셀이 편집됩니다. 설정 → 데이터 편집기:

- **휠을** 사용하여 열을 스크롤하거나 행을 스크롤할 수 있습니다.
- **색상** — 그리드, 헤더 및 선택 영역.

자산

ASSETS는 전체 프로젝트가 아닌 현재 열려 있는 부분만 가져오고 내보냅니다. 스프라이트는 PAINT에서 열려 있는 시트의 스프라이트를 사용합니다. 맵은 현재 열려 있는 맵입니다. SFX는 현재 열려 있는 패턴입니다. 음악은 현재 열려 있는 곡입니다. 데이터는 현재 열려 있는 테이블입니다. 코드는 예외입니다. 모든 탭이 작성되고 모든 탭이 다시 읽어집니다.

친절한	파일	움직이는 것
스프라이트	`.png`	열린 스프라이트.
지도	`.p37map`	열린 지도.
효과음	`.p37sfx`	개방형 패턴.
음악	`.p37mus`	시작 곡입니다. 패턴 번호는 그대로 유지됩니다. 효과음이 필요한 경우 별도로 내보내세요.
데이터	`.p37dat`	열린 테이블.
암호	`.lua`	모든 탭.

설정

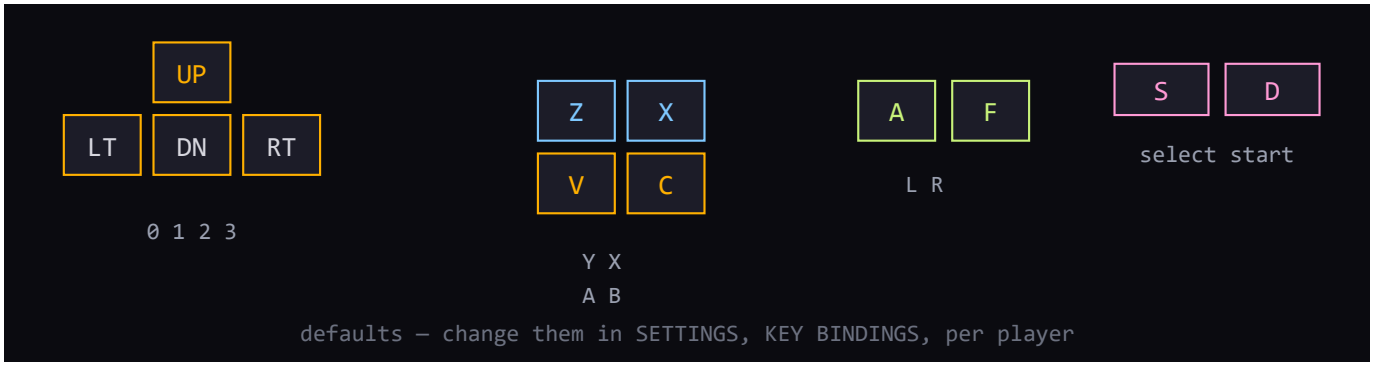
루트 목록:

페이지	저장하는 내용
프로필	모든 설정을 저장하고 불러올 수 있습니다. 새로 만들기, 이름 변경, 덮어쓰기, 삭제가 가능합니다. 프로필을 텍스트로 복사하거나 붙여넣을 수 있습니다.
메뉴	창, 키, 메뉴 색상, 선택적 배경.

게임 패널	미리보기 및 재생 창.
코드 편집기	강조 표시 및 힌트.
페인트 에디터	브러시, 팔레트, 지도 보기.
SFX 에디터	신디사이저 및 음표 입력.
데이터 편집기	휠 방향 및 색상.
언어	메뉴 및 힌트 언어.
라이선스	라이선스 텍스트에는 Fusion Pixel 글꼴 및 SDL3가 포함됩니다.

글로벌:

- **레이아웃** — 패널 배치도(위 참조).
- **볼륨** — 마스터 레벨, 0~10.
- **키 바인딩** — 8명의 플레이어 각각에 대한 키입니다. 키를 눌러 해당 키를 선택하세요. 방향, ABXYLR, 선택.
- **현재 상태** — 유휴 프레임을 건너뛰거나 모든 프레임을 그립니다.
- **전체 화면** - 테두리 없음.
- **새로 고침** - 창 FPS. 왼쪽 및 오른쪽 단계는 60, 90, 120, 최대값입니다. Enter 키를 눌러 최대 1000까지 숫자를 입력하세요. 0은 제한 없음입니다.
- **Vsync** — 프레임 간 화면 표시를 기다립니다.
- **배율** — 정수 윈도우 승수.
- **셰이더** - CRT 및 필터는 게임 자체가 아닌 에디터에서 적용됩니다.
- **색상** - 메뉴 텍스트 및 강조 표시.
- **배경** — 메뉴 뒤에 표시되는 선택 사항인 PNG 이미지(크기 128). [지우기]를 누르면 제거됩니다.
- **편집기 커서** - 포인터 모양.
- **마우스** - 플로팅 포인터 또는 픽셀에 고정된 포인터.
- **UI 속도** - 길게 누른 방향이 목록을 통해 반복되는 속도.
- **탐색기** — 파일 목록은 컴퓨터의 루트 폴더, 즉 실행 파일이 있는 폴더에서 시작됩니다.



기본 키보드 설정입니다. A는 메뉴에서 확인 버튼이고, B는 뒤로 가기 버튼입니다. 게임에서는 4번과 5번 버튼이 해당합니다. 선택(S)은 10번 버튼이며, 메뉴 단축키는 아닙니다.

단추	역할	기본 키
0	왼쪽	왼쪽 화살표
1	오른쪽	오른쪽 화살표
2	위로	위쪽 화살표
3	아래에	아래쪽 화살표
4	A, 확인	다섯
5	B, 뒤로	기음
6	엑스	엑스
7	와이	지
8	아르 자형	에프
9	엘	에이
10	선택하다	에스
11	시작	디

짓다

플랫폼을 선택하세요: Windows, Linux, Mac, Android, iOS 또는 웹.

- **이름** — 빌드의 파일 이름입니다.
- **작성자** - Windows 및 Linux 사용자. 이것은 저장된 파일이 저장되는 폴더 이름입니다.
- **Steam 앱 ID** — Windows 및 Linux에서. 비어 있으면 Steamworks가 없음을 의미합니다. 채워진 ID는 `achieve(0)` … `achieve(15)` 를 Steam 도전 과제 ACH_0…ACH_15에 매핑합니다.

- **ID** — 플랫폼에서 요청하는 경우 번들 식별자입니다.
- 데스크톱 및 모바일 오버레이에서 **코드 숨기기 기능을 제공합니다** . 게임 카트리지 미리 `peek` 늦춰주지만, 잠금 기능은 아닙니다.
- **전체 화면** — 내보낸 플레이어가 전체 화면으로 시작됩니다.
- **셰이더** - 플레이어에 최대 3개의 레이어가 적용됩니다.
- **.p37 파일 다운로드** — 웹 전용. 일시 정지 메뉴에 다운로드 항목이 표시됩니다.
- **아이콘** — 스프라이트 시트의 타일: 시트, 크기, x, y. 또는 PNG 커버 이미지를 잘라낼 수도 있습니다.
- **내보내기** — 폴더를 선택하세요. 메뉴에 빌드 저장됨 또는 빌드 실패 여부가 표시됩니다.

인터넷이 연결된 상태에서 빌드를 생성하세요. 실행 화면에서 서버에 해당 빌드의 해시값이 알려져 있는지 익명으로 문의할 수 있습니다. 이러한 확인 절차는 두 가지 이유로 필요합니다. 첫째, PIX37의 제작자인 저를 무단 복제로부터 보호하기 위함입니다. 둘째, 게임 제작자를 보호하기 위한 것입니다. 다른 사람의 게임을 가져다가 자신의 것처럼 등록하는 행위가 Play 스토어(Android)나 iOS와 같은 모바일 앱 스토어에서 흔해지고 있습니다. 게임은 정상적으로 실행됩니다. 기기가 인터넷에 연결되어 있지 않은 경우, 게임 시작 화면에 "게임이 연결되어 있지 않습니다"라는 메시지가 표시됩니다. 이 메시지는 플레이어에게 해당 복제본이 불법적인 경로로 배포되었을 가능성을 알려주는 역할을 합니다. 또한, 누군가가 당신의 게임을 무단으로 도용하여 판매했을 경우, Play 스토어(Android), Steam, itch.io와 같은 스토어에서 이의를 제기할 수 있도록 하는 간단한 절차입니다. 서버는 해시값 옆에 최초 등록 시간을 저장하고 이후에는 변경하지 않습니다. 최초 등록 시간은 오리지널 빌드이고, 이후에 등록된 해시값은 이후 복제본입니다.

저는 이 검사가 게임 개발자들을 보호하기 위한 것이라고 생각합니다. 게임 엔진이 이미 많은 부분을 대신 해주고 있고, 이것도 그중 하나입니다. 논란의 여지가 있을 수 있다는 점은 이해합니다. 만약 커뮤니티에서 이 마이크로 DRM을 없애는 것이 더 낫다고 판단한다면, 그렇게 하겠습니다.

이전 에디터에서 사용하던 게임 카트리지가 언어 변경 후 부팅되지 않을 수 있습니다. 이 에디터에서 `.pix37` 프로젝트를 열고 새 `.p37` 파일로 내보내세요.

PIX37.PARTY에 접속하세요

PIX37.PARTY에 접속하려면 앱 내에 로그인해야 합니다. 브라우저에는 로그인 페이지가 없습니다. Steam 세션 티켓 또는 itch.io 계정을 사용하여 패드와 마우스로 로그인합니다. 서비스는 2~16자의 고유한 표시 이름을 보여줍니다. 라이선스 키와 Steam ID는 다른 플레이어에게 표시되지 않습니다.

서비스 약관을 읽고 나면 즐겨찾기, 카탈로그, 순위, 플레이어, 그리고 프로필이 메인 화면에 나타납니다.

개발자 라이선스는 에디터입니다. 게임을 만들고 업로드한 후 별점을 남길 수 있습니다. 게이머 라이선스는 플레이어입니다. 하트는 러너 게임을 플레이하는 사람들의 평점입니다.

- **카탈로그** — 게임을 찾아보세요. 장르(최대 3개), 화면 크기, 게임 카트리지 크기, 그리고 업로드에 `.pix37` 프로젝트가 포함되어 있는지 여부를 필터로 설정할 수 있습니다.
- **즐겨찾기** — 표시해 둔 게임입니다.
- **업로드** — 개발자 라이선스가 필요합니다. `.p37` 파일을 전송하고, `.pix37` 프로젝트 파일을 첨부할 수 있습니다. `.p37` 파일은 압축된 바이트코드입니다. 읽기가 어렵고, 의도적으로 난독화될 수도 있습니다.

`.pix37` 파일은 실제로 작업하는 프로젝트 파일입니다. 소스 코드, 그림, 사운드 등이 포함되어 있으며, 다른 사람이 열어서 학습할 수 있는 형식입니다. 이 파일을 전송하면 다른 제작자들이 압축된 게임 카트리지 대신 게임 자체를 연구할 수 있습니다. 제목, 제작자, 장르, 표지, 아이콘, 업적 이름 등의 정보도 함께 전송됩니다. 전송되는 RAM 용량은 게임 카트리지가 도달한 최대치입니다.

- **댓글 및 업적** 은 게임 페이지에서 확인할 수 있습니다. 로그인 상태에서 업적을 달성하면 해당 계정과 기기에 모두 저장됩니다.
- **순위** - 개발자를 별점 또는 하트 개수 순으로 정렬합니다. 각 행에는 두 가지 점수가 모두 표시됩니다. 이름을 클릭하면 해당 개발자의 게임 목록을 하트 개수 순으로 볼 수 있습니다.
- **플레이어** - 달성한 업적 순으로 정렬된 계정 목록입니다.

암호

이 언어는 간단한 Lua 스타일 스크립트입니다. 이 부분에서는 빈 프로젝트에서 시작하여 언어를 가르치고 각 기능의 간단한 예제를 보여줍니다.

첫 번째 프로그램

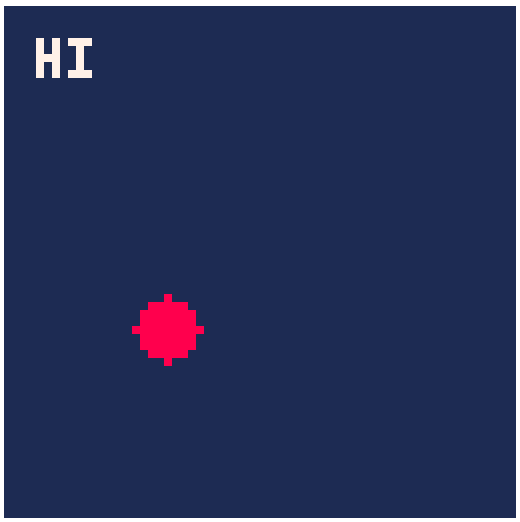
64x64 해상도, 4비트 색상으로 프로젝트를 생성합니다. CODE를 엽니다. 탭을 다음으로 바꿉니다.

```
_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 키를 누르면 게임이 실행됩니다. 화살표 키로 원을 이동시킬 수 있습니다. Esc 키를 누르면 게임이 종료되고 편집기로 돌아갑니다.



동일한 세 가지 호출: `cls(1)` 은 남색 화면을 그리고 `print` 는 HI를 그리고 `circfill` 분홍색 원을 그립니다.

무엇이 실행되고, 언제 실행되나요?

- `_init` 한 번 실행됩니다.
- `_update` 60Hz로 실행됩니다. 여기에 동작 및 버튼을 배치하세요.
- `_draw` 그림을 그립니다. fps 인수가 없으면 컴퓨터와 모니터가 처리할 수 있는 최대 속도로 그림을 그립니다. `_draw(30)` 은 그림 그리기를 30fps로 제한하고 논리는 60fps로 유지합니다.
- `_set_fps(n)` 나중에 해당 제한을 변경합니다. 0은 제한을 해제합니다.

이름은 대소문자를 구분하지 않으며, 편집기는 소문자로 저장합니다. `function` 과 `fun` 은 동일한 키워드입니다.

언어

프로그래밍 기초

프로그램은 게임 카트리지가 실행하는 일련의 코드 라인입니다. 각 제목은 먼저 해당 라인의 형태를 보여주고, 그 다음 간단한 예제를 보여줍니다. `--` 로 시작하는 라인은 주석, 즉 사용자를 위한 안내 사항입니다. 게임은 이 라인을 건너뜁니다.

변수

변수는 값을 저장하는 이름입니다. 이름은 여러분이 정할 수 있습니다. 이후의 코드에서 이 이름을 읽어 같은 이름에 새로운 값을 저장할 수 있습니다.

형태는 `name = value` . `local name = value` 해당 이름을 현재 블록 내에 유지하므로 게임 카트리지의 다른 탭에서는 해당 이름을 볼 수 없습니다. `local` 없으면 이름은 전역적으로 적용되어 모든 탭에서 볼 수 있습니다.

```
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` 값이 없음을 의미합니다. `true` 와 `false` 는 각각 '예'와 '아니오'를 의미합니다. 한 번도 설정하지 않은 이름은 `nil` 로 표시됩니다.

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

테이블

테이블은 하나의 이름 아래 여러 값을 저장합니다. 리스트는 숫자를 사용합니다. 첫 번째 항목은 0이 아니라 1입니다. 레코드는 이름을 사용합니다. 테이블은 다른 테이블을 포함할 수 있습니다.

리스트 형태: `{ value, value }` . 레코드 형태: `{ name = value, name = value }` . 레코드 리스트는 항목이 테이블인 테이블입니다.

```
-- 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
```

상태 및 신체

조건은 `hp > 0` 또는 `btnd(4)` 와 같이 예 또는 아니오로 답할 수 있는 질문입니다. 본문은 블록에 진입할 때 실행되는 줄 그룹입니다. `if` 와 모든 루프는 동일한 모양(조건, 본문, `end` 을 사용합니다. `repeat` `end` 대신 `until` 로 끝납니다.

만약에

`if` 조건문은 조건이 참일 때만 본문을 실행합니다. `elseif` 조건문은 다른 조건을 시도합니다. `else` 조건문은 모든 조건이 거짓일 때 실행됩니다.

형태: `if condition then` 본문, 그 다음 `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
```

루프

`loop`는 본문을 반복 실행합니다. `while` 먼저 조건을 확인하고 조건이 참인 동안 계속 실행합니다. `repeat` 본문을 실행한 후 조건을 확인하므로 본문은 항상 최소 한 번 실행됩니다. `for` 횟수를 세는데, 마지막 숫자도 포함됩니다. 세 번째 숫자가 없으면 횟수는 1이 되므로 시작점이 이미 끝점을 지나치면 본문이 실행되지 않습니다. 감소 카운트를 하려면 음수 횟수가 필요합니다.

```
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
```

테이블 위를 걷다

`all` 리스트의 각 항목을 첫 번째 항목부터 차례로 방문합니다. `pairs` 각 이름과 그 이름 아래에 저장된 값을 방문합니다.

```
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
```

함수

함수는 이름이 붙은 레시피입니다. 형식은 `function name(arguments)` , 본문, 그리고 `end` . 함수는 이름과 괄호를 써서 호출합니다. `return` 값을 반환하며, 두 개 이상의 값을 반환할 수도 있습니다. 인수는 해당 함수의 지역 변수입니다.

```
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` 는 주변의 로컬 변수들을 볼 수 있는 함수입니다. 이것이 바로 클로저입니다. 테이블이 로컬 함수를 담을 수 있고, 점(.)을 붙여서 호출합니다.

```
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
```

예, 아니오, 그리고 선택하기

`not` 조건을 뒤집습니다. ` `and` 양쪽 모두 참일 때만 참입니다. ` `or` 어느 한쪽이 참이면 참입니다. `a and b or c` ` `a` 참일 때 `b` 선택하고, 그렇지 않으면 `c` 선택합니다. `cond ? a : b` 는 같은 개념을 한 줄로 표현한 것으로, 조건이 참이면 `a` 선택하고, 거짓이면 `b` 합니다.

```
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)
```

수학과 텍스트

`+ - * /` 는 각각 덧셈, 뺄셈, 곱셈, 나눗셈을 나타냅니다. `//` 는 나누기를 하고 나머지는 그대로 둡니다. `%` 는 나머지입니다. `^` 는 거듭제곱입니다. `..` 는 두 텍스트를 붙입니다. `#t` 는 리스트의 항목 개수입니다. 비교 연산자는 `==` (같음), `~=` 또는 `!=` (다름), 그리고 `< > <= >=` . `x += 1` x에 1을 더합니다. `-= *= /= ^= %=` 같은 방식으로 비교합니다.

```
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
```

현지인이 사는 곳

`local` 해당 블록을 닫는 `end` 까지 유지됩니다. `if` ` `end` 문은 그 자체로 하나의 블록입니다. 변수를 설정하는 줄 바로 위에 `local` 변수를 언급할 수는 있지만, 해당 줄이 실행되기 전까지는 값이 `nil` 입니다.

```
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
```

점프

`::label::` 은 줄바꿈을 표시합니다. `goto label` 해당 줄로 이동하지만, 같은 함수 내부에서만 가능합니다.

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

간단히 말해서, 기억력

`peek` 과 `poke` 16비트 주소 하나를 통해 실시간 픽셀을 살펴봅니다. 게임 카트리지의 두 번째 복사본을 저장하지 않습니다. 주소 `0x5F00` 부터 `0x5FFF` 까지는 카메라, 클립(읽기 전용), 그리기 색상, 채우기 패턴, 세 가지 팔레트 재매핑 및 페이지 번호 레지스터입니다. 해당 주소 양쪽의 65280바이트는 픽셀 창입니다.

화면이 65280바이트 공간에 맞으면(64x64 및 128x128 크기는 맞지만 256x256 크기는 맞지 않음), 화면은 주소 0부터 행 단위로 표시됩니다. 남은 공간은 `0x5F41` 로 선택된 스프라이트 시트입니다(0은 시트 0). 해당 시트가 남은 공간보다 크면 `0x5F42` 페이지를 선택하고 맵은 창에 표시되지 않습니다. 시트가 공간에 맞으면 `0x5F43` 으로 선택된 맵의 셀(각 1~2바이트)이 꼬리 부분에 표시되고, 꼬리 부분에도 맞지 않으면 `0x5F44` 맵을 페이지별로 넘깁니다.

화면이 맞지 않을 경우 전체 창은 65280픽셀 크기의 한 페이지입니다. `poke(0x5F40, n)` 누르면 화면 페이지가 넘어갑니다. n이 0이 아닌 경우 `poke(0x5F41, n)` 누르면 시트 n이 표시되고, 이 모드에서는 시트 0에 뷰가 없습니다. n이 0이 아닌 경우 `poke(0x5F43, n)` 누르면 맵 n이 표시되고, 이 맵이 시트보다 우선합니다. 이 모드에서 맵 0에는 `mget` 과 `mset` 만 표시됩니다.

픽셀이나 레지스터에 닿지 않는 `peek` 작업은 0을 반환합니다. 해당 위치를 `poke` 아무런 동작도 하지 않습니다. 스프라이트 플래그, SFX 및 DATA 테이블은 이 창에 표시되지 않습니다. `fget` , `sfxget` 및 `dget` 사용하십시오.

실용적인 용도

이 페이지 하단에는 모든 기능이 나열되어 있으며, PIX37 기능의 실제 사용 예시가 나와 있습니다. 각 예시 아래의 링크를 클릭하면 해당 기능에 대한 자세한 정보를 확인할 수 있습니다.



부드러운 움직임

위에서 본 128×128 크기의 게임 카트리지입니다. 스프라이트 1이 이미 그려져 있습니다. 화살표는 모든 방향으로 가속하며, 마찰력은 속도를 감소시킵니다. 스프라이트는 감싸는 형태로 그려지는데, 아래쪽에서 벗어나는 절반은 위쪽에 그려지고, 나머지 가장자리도 마찬가지입니다.

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



다음 수를 둘 곳이 막혀 있는지 확인합니다.

64x64 크기의 게임 카트리지입니다. 바닥과 오른쪽 벽은 이미 스프라이트 3으로 표시되어 있으며, 깃발 0이 해당 스프라이트 위에 놓여 있습니다. 화살표는 스프라이트 1을 이동시킵니다. 상단의 단어는 다음 이동 경로가 단색 타일과 겹칠 경우 '차단됨', 겹치지 않을 경우 '개방됨'으로 표시됩니다.

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



뒤집을 수 있는 물티슈

A 키를 누르세요. 화면 지우개 기능은 _draw() 함수 내부가 아닌 flip() 함수에 의해 검은색 행이 하나씩 그려집니다. _draw 함수는 flip() 함수가 실행되는 동안 방해받지 않으므로 각 행은 화면에 계속 표시됩니다.

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



함수 표 (클래스)

스프라이트 1은 이미 그려졌습니다. make_monster는 클래스처럼 작동합니다. 테이블을 반환하며, update와 draw 메서드는 해당 몬스터의 x, y 좌표를 유지합니다. 맨 위에 있는 숫자는 몬스터가 이동한 거리입니다.

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



버튼을 읽는 네 가지 방법

A는 버튼 4입니다. 누르고 있으면 충전량이 증가하고, 놓으면 충전량이 0으로 돌아갑니다. 누르고 있는 동안 위아래 버튼도 같은 동작을 반복하므로 메뉴 행이 계속 이동합니다. btnd는 버튼을 누르는 첫 번째 프레임으로, 이 게임 카트리지에서는 누르는 동작의 시작으로만 사용됩니다.

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



마우스로 쏘세요

스프라이트 1이 이미 중앙에 그려져 있습니다. 흰색 점은 커서입니다. 왼쪽 클릭은 해당 점을 향해 픽셀을 발사합니다. 휠을 돌리면 속도가 변경되며, 속도는 화면 모서리에 표시됩니다.

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



이름을 입력하고 확인하세요

A 키를 눌러 입력란을 여세요. 위아래 화살표 키로 깜빡이는 글자를 바꿀 수 있습니다. A는 입력, X는 삭제, Y는 공백, B는 확인입니다. 입력이 완료되면 이름이 표시됩니다.

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



플레이어의 카메라

128×128 크기의 게임 카트리지입니다. 스프라이트 1은 타일 바닥의 중앙에서 시작합니다. 화살표가 움직입니다. 카메라는 스프라이트를 중앙에 유지합니다. 모서리의 x, y 좌표는 스크롤되지 않는데, 이는 `camera()` 함수에 인수가 없으면 HUD를 위해 시점이 뒤로 이동하기 때문입니다.

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



비주얼 노벨의 몇 줄

대사는 DATA 형식으로 영어와 폴란드어로 되어 있습니다. A는 계속 진행하거나, 선택지가 화면에 나타날 때 '유지'를 선택합니다. B는 선택지가 나타나면 '떠나기'를 선택하고, 그렇지 않으면 언어를 바꿉니다. X는 선택지가 표시되는 동안 언어를 바꿉니다. 대화 상자는 8px 글꼴을 사용하므로 폴란드어 글자가 잘 보입니다.

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



엿보고 찌르면서 그림을 그리세요

64x64 게임 카트리지에서 주소 0은 왼쪽 상단 픽셀이고 다음 행은 64부터 시작합니다. 게임 카트리지는 픽셀을 기록하고, 한 행을 복사하고, 한 행을 지우고, 읽어온 값을 출력합니다.

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



추가할 수 있는 목록

A는 8까지의 숫자를 더하고, B는 마지막 숫자를 빼고, X는 첫 번째 숫자를 뺍니다. 목록이 화면에 출력됩니다.

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



이름이 분해되었습니다

게임 카트리지는 ada,12,blue를 분할하여 이름, 첫 글자, 12, 그리고 blue라는 단어를 인쇄합니다.

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



음을 연주한 다음, 음을 바꾸세요.

패턴 0은 한 음표 길이이며, 재생될 때마다 음량이 동일합니다. A는 해당 음이 끝나면 다시 연주합니다. B는 음높이를 높여서 연주합니다.

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



게임 카트리지에 적힌 노래

0번째 행에서 두 개의 패턴이 반복 재생되는데, 각각 첫 번째와 두 번째 채널에 하나씩 적용됩니다. 행 번호는 0으로 유지됩니다. ch0과 ch1은 각 채널에 설정된 음표입니다. 게임 카트리지에서 패턴, LEN, LOOP 값을 편집하면 음악 패널에서 해당 컨트롤이 어떻게 작동하는지 확인할 수 있습니다.

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



DATA에 있는 세 마리의 괴물

데이터 패널에는 이미 슬라임, 박쥐, 눈이라는 이름 열과 HP 열이 있습니다. A는 슬라임을 공격합니다. 오른쪽에 있는 숫자는 슬라임의 HP입니다.

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



점수를 기억하세요

A는 점수를 추가하고 저장합니다. 게임을 종료하고 게임 카트리지를 다시 열면 점수가 그대로 남아 있습니다. B는 그 점수를 잊어버립니다.

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




다른 파일의 두 번째 장

두 개의 게임 카트리지가 저장 슬롯 퀘스트를 공유합니다. 파트 1: A는 게임 완료를 나타냅니다. 먼저 열리는 파트 2에서는 파트 1을 완료하라는 메시지가 표시됩니다. 그 후 파트 2가 열리고, B는 저장 내용을 지워 파트 1을 다시 플레이할 수 있도록 합니다.

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



하나의 업적

스프라이트 8이 이미 그려졌습니다. A 버튼을 10번 누르세요. 토스트 메시지에 업적 이름이 표시되고, 스프라이트 8은 잠금 해제되었으므로 그대로 유지됩니다. B 버튼을 누르면 로컬 파일에서 해당 업적이 삭제되고 카운트가 초기화됩니다.

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



들어갈 수 있는 방

256x256 크기의 게임 카트리지입니다. 흰색 점이 커서입니다. 메뉴는 호스트, 참가, 서버 목록으로 구성되어 있습니다. 호스트는 'walk'라는 이름의 방을 열고 게임을 시작합니다. 참가 버튼을 누르면 주소(숫자, 점, 콜론)가 스크롤되어 표시되고, B가 접속합니다. 서버 목록은 위아래로 움직이며, A가 해당 행에 참여합니다. 게임 화면에서 화살표 키는 첫 번째 스프라이트를 이동시킵니다. 동전 하나는 공유됩니다. 호스트가 동전을 이동시키고, 플레이어가 동전을 집으면 해당 플레이어의 점수가 1점 증가하고 다음 동전이 생성됩니다. 모든 스프라이트(자신의 스프라이트 포함) 옆에는 점수가 표시됩니다. 모서리에는 현재 참여 중인 플레이어 수, LAN 주소, 호스트의 공용 IP 주소가 표시됩니다. B가 나가면 자신과 다른 플레이어들은 메뉴로 돌아가고, 다음 호스트가 점수 0에서 새로운 방을 시작합니다.

```
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
```

 기다리는 대화

A 버튼을 누르면 다음 줄이 표시됩니다. 함수는 A 버튼을 다시 누를 때까지 각 줄바꿈 지점에서 일시 정지됩니다.

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

 잠깐만 기다려 주세요

단어가 약 0.5초 후에 "..."에서 "hi"로 바뀝니다. 코루틴이 일시 중지된 동안에도 화면은 계속 그려집니다.

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

 하나씩 도착하는 원들

다섯 개의 원이 위쪽에서 두 프레임 간격으로 떨어져 내립니다.

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

 시계에서 가져온 두 개의 프레임

스프라이트 1과 스프라이트 2는 동일한 캐릭터의 두 프레임입니다. 이 그림은 초당 네 번 바뀝니다.

```
cls flr time spr _draw
```

모든 기능

대괄호 안의 인수는 선택 사항입니다. 테이블과 문자열의 인덱스는 1부터 시작하지만, DATA 행과 열의 인덱스는 0부터 시작합니다. 색상은 팔레트 인덱스입니다.

색인

_그리기
_init
_mp
_mp_fail
_mp_join
_mp_leave
_mp_update
_set_fps
_업데이트
복근
achclear
achdef
달성하다
달성
추가하다
모두
주장하다
atan2
밴드
아니
보르
btn
btnd
btnp
btnu
bxor
카메라
천장
문자
원
둘러싸다
클릭
클릭
클리커
클립
cls
공동 창조
색깔
코어수메
왜냐면
코스타투스
카운트
커서
DCOL
델
델리
저장 취소
dget
졸다
d세트
dstr
fget
채우다
뒤집기
flr
foreach
fset
정보
아이페어
키오프
키온
키스텟
키스트
라인
부하
로그
lshr
지도
최대
밈피
memset
mget
중반
분
쥐
mset
음악
음악통
발표하다
전화
다음
nhost
가입하지 않음
nkb
nkbstr
란
떠나다
목록
들어봐
참고
nplayers
nsend
nstun
질서
타원형
타원형 채우기
팩
쌍
친구
palrgb
팰트
PC 통화
엿보다
엿보기2
엿보기4
성능
pget
찌르다
포케2
포케4
인쇄
인쇄10
인쇄8
pset
직사각형
바로 채우기
런드
로틀
회전
바로잡다
수정하다
저장
저장 슬롯

스크린
스크롤
선택하다
세트맵
세트 팔레트
세트프르
세텍스
좌절
SF
sfxget
sfxhdr
sfx세트
sfxstat
겟
sgn
shl
쇼 통계
쉬르
죄
송겟
노래 세트
분할
봄
sqrt
모래
세트
sspr
통계
서브
시간
라인
토넘
tostr
포장을 풀다
후아미
수확량

생명주기

_init

`_init() ... end`

게임 카트리지가 시작될 때 한 번 실행됩니다. 어떤 훅에도 속하지 않는 코드는 모두 _init 함수에 포함됩니다.

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

_update

`_update() ... end`

게임 로직. _draw가 제한되거나 호스트 속도가 느리더라도 60Hz로 호출됩니다. 버튼 읽기 기능이 여기에 있습니다.

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

_draw

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

한 프레임을 그립니다. 인수가 없거나 0이면 호스트가 표시하는 빈도만큼 실행됩니다. 초당 n프레임으로 그리기를 제한하려면 n(1..999)을 전달하십시오. _update는 60Hz로 유지됩니다.

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

_set_fps

```
_set_fps(n)
```

게임 실행 중 _draw 제한 값을 변경합니다. 0은 제한 없음, n은 0~999로 제한됩니다.

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

flip

```
flip()
```

화면을 표시하고 다음 프레임이 나타날 때까지 대기합니다. 프레임 루프를 직접 그리는 경우에만 사용하십시오. 이후에는 대기하지 마십시오.

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

time

```
time()
```

_init 이후 경과된 초.

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

입력

btn

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

버튼 b를 누르고 있는 동안에는 1, 그렇지 않으면 0입니다. p는 플레이어이며 0부터 7까지의 값을 가집니다. 인수가 없으면 플레이어 0의 비트 마스크를 반환합니다. 버튼: 0 왼쪽, 1 오른쪽, 2 위, 3 아래, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 선택, 11 시작.

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

btnd

`btnd(b, [p])`

버튼 b를 누르는 _업데이트 프레임에서만 참입니다.

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

btnp

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

버튼을 누르면 참이 되고, 누르고 있는 동안에는 30틱의 지연 후 8틱마다 다시 참이 됩니다. 버튼을 누르지 않으면 마스크를 반환합니다.

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

btnu

`btnu(b, [p])`

버튼이 놓이는 프레임에서만 참입니다.

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

mouse

```
mouse()
```

마우스의 x, y 좌표를 게임 픽셀 단위로 반환합니다.

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

clickl

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

왼쪽 마우스 버튼. clickl()이 눌러진 상태입니다. clickl(1)은 누르는 가장자리입니다.

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

clickr

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

마우스 오른쪽 버튼. clickl과 동일한 1개 인수 엣지.

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

clickm

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

가운데 버튼을 누르거나 마우스 휠을 클릭하면 됩니다.

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

scroll

`scroll()`

마우스 휠의 움직임은 이 프레임에서 긍정적 또는 부정적입니다.

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

keyon

`keyon()`

버튼으로 조작하는 텍스트 필드를 엽니다. 위아래 버튼으로 A~Z 문자를 선택할 수 있습니다. A는 해당 문자를 입력하고, X는 삭제하고, Y는 공백입니다. B는 필드를 확정하고 닫습니다. 키보드로는 입력할 수 없습니다.

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

keyoff

`keyoff()`

텍스트 입력란을 숨깁니다.

```
keyoff()
```

keystr

`keystr([s])`

입력 필드에 텍스트를 입력한 다음 프레임 B의 1을 눌러 확인합니다. 그런 다음 위아래로 글자가 바뀝니다. 필드가 닫히면 해당 글자는 비어 있게 됩니다. keystr("")는 텍스트를 지웁니다.

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

keystat

`keystat()`

텍스트 입력란이 열려 있는 동안에는 1, 그렇지 않으면 0입니다.

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

그림

screen

`screen()`

현재 게임 카트리지 화면 크기(픽셀)입니다. 너비를 반환한 다음 높이를 반환합니다. 이는 게임 해상도이며, 에디터 창의 해상도가 아닙니다.

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

cls

`cls([col])`

화면을 지웁니다. 기본 색상은 0입니다.

```
cls(1)
```

color

`color(col)`

이후 호출에서 `col` 매개변수가 생략될 경우 사용될 그리기 색상을 설정합니다. 매개변수 없이 호출하면 현재 색상을 반환합니다.

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

camera

`camera([x, y])`

모든 드로우 콜마다 오프셋을 적용합니다. camera() 함수는 0, 0으로 재설정됩니다.

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

clip

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

그리기 영역을 사각형으로 제한합니다. 0이 아닌 다섯 번째 인수는 현재 클립 영역과 교차합니다. clip() 함수는 전체 화면을 복원합니다. 클립 레지스터의 미리보기는 읽기 전용입니다.

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

cursor

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

좌표 없이 다음 출력을 표시할 위치와 선택적으로 색상을 설정합니다. cursor()는 x, y 값을 반환합니다.

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

pset

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

화면 픽셀 하나를 설정합니다. 픽셀 통계에 포함됩니다.

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

pget

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

화면의 x, y 위치에 있는 픽셀의 색상입니다.

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

print

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

3×5 크기의 텍스트(UTF-8, Fusion 폰트에서 잘라낸 부분)를 그립니다. x, y 값을 지정하지 않으면 커서 위치에 텍스트를 출력하고 다음 줄로 이동합니다.

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

log

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

인수들을 공백으로 구분하여 텍스트 버퍼에 추가합니다. pix37_run은 해당 버퍼 내용을 표준 출력으로 출력합니다. 그림은 그리지 않습니다.

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

print8

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

인쇄물과 동일하며, 8px Fusion 글꼴을 사용했습니다.

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

print10

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

인쇄물과 동일하며, 10px 크기의 Fusion 글꼴이 사용되었습니다.

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

line

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

x0,y0에서 x1,y1까지의 선입니다. 두 개의 인수는 이전 끝에서 이어집니다. line() 함수는 해당 연결 고리를 초기화합니다.

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

circ

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

원형 윤곽선. 반지름은 픽셀 단위입니다.

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

circfill

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

채워진 원.

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

oval

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

경계 상자 내부에 타원 윤곽선이 그려져 있습니다. 모서리는 모두 포함됩니다.

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

ovalfill

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

경계 상자 안에 채워진 타원.

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

rect

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

직사각형 윤곽선. 모서리도 포함됩니다.

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

rectfill

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

채워진 직사각형.

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

rrect

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

모서리가 둥근 직사각형 윤곽선입니다. x, y는 왼쪽 상단 좌표입니다. w와 h는 픽셀 단위의 크기이며, 둘 다 0보다 커야 합니다. r은 모서리 반지름이며 기본값은 0입니다. 짧은 변의 절반으로 제한됩니다.

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

rrectfill

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

채워진 모서리가 둥근 사각형입니다. rrect와 동일한 인수를 사용합니다.

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

fillp

`fillp([p])`

나중에 채워질 도형에 사용할 16비트 채우기 패턴입니다. 0은 채워진 상태를 나타냅니다. fillp() 함수는 현재 패턴을 반환합니다. 비트는 4×4 패턴을 따라 이동합니다.

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

pal

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

색상 c0을 c1으로 다시 매핑합니다. 모드 0은 그리기 팔레트, 1은 표시 팔레트, 2는 채우기 패턴 팔레트입니다. pal()은 초기화합니다.

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

palrgb

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

팔레트 항목 하나를 RGB로 설정하고, 각 채널의 값을 0~255로 지정합니다.

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

palt

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

투명도. palt(col) 또는 palt(col, true)는 col을 투명하게 만듭니다. palt(col, false)는 불투명하게 만듭니다. palt()는 모든 색상을 불투명하게 만듭니다.

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

setpalette

```
setpalette(n)
```

팔레트 뱅크 n을 선택합니다. setpalette()는 현재 뱅크를 반환합니다.

```
setpalette(1)
```

tline

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

지도 위치 mx, my에서 지도 픽셀을 샘플링하면서 화면에 선을 그립니다. mdx, mdy는 선을 따라 지도 샘플링 단계를 지정합니다.

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

스프라이트와 맵

spr

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

스프라이트 n을 x, y 위치에 그립니다. w와 h는 픽셀 단위의 크기이며 기본값은 8입니다. fx와 fy는 0이 아닐 경우 가로 및 세로로 뒤집습니다. tex는 스프라이트 시트 탭입니다. rot는 회전 각도(0~1)이며 sin 및 cos와 같은 방향입니다. 0으로 설정하면 스프라이트가 회전하지 않습니다.

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

sspr

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

스프라이트 시트의 직사각형 영역을 화면에 복사하고, 크기는 dw x dh로 조정합니다. 기본 설정은 원본 크기를 유지합니다.

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

sget

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

스프라이트 시트의 한 텍셀의 색상입니다.

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

sset

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

스프라이트 시트에 텍셀 하나를 그립니다.

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

fget

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

스프라이트 플래그. fget(n)은 플래그 바이트를 반환합니다. fget(n, f)는 플래그 비트 f(0..7)가 설정된 경우 참입니다.

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

fset

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

스프라이트 플래그를 설정합니다. fset(n, v)는 전체 바이트를 기록합니다. fset(n, f, v)는 한 비트를 설정합니다.

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

setspr

```
setspr(n)
```

활성화된 스프라이트 시트를 선택합니다. setspr()은 현재 시트를 반환합니다. settex와 동일한 뱅크를 사용합니다.

```
setspr(1)
```

settex

```
settex(n)
```

setspr의 별칭.

```
settex(0)
```

map

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

지도를 그립니다. 지도의 셀 cx, cy는 화면 sx, sy 위치에 그려집니다. cw, ch 값이 0이면 전체 지도를 의미합니다. 레이어는 허용되고 무시됩니다. tex는 타일셋 시트입니다.

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

mget

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

스프라이트 ID는 맵 셀 cx, cy에 있습니다.

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

mset

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

스프라이트 n을 맵 셀 cx, cy에 기록합니다.

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

setmap

```
setmap(n)
```

맵 n을 선택합니다. setmap()은 현재 맵을 반환합니다. 맵 0은 실수 맵입니다.

```
setmap(1)
```

메모리

peek

`peek(addr)`

64KB 창에서 1바이트를 읽습니다. 창은 라이브 화면, 스프라이트 픽셀, 그리고 화면이 65280바이트에 맞을 경우 맵 셀 순으로 구성됩니다. 더 큰 화면은 poke(0x5F40, page)를 사용하여 페이지네이션됩니다. 매핑되지 않은 주소는 0을 반환합니다. @addr로도 표기됩니다.

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

poke

`poke(addr, ...)`

주소에 바이트를 기록합니다. 추가 인수를 사용하면 다음 바이트가 기록됩니다. 매핑되지 않은 주소는 무시됩니다. $addr, val도 함께 기록됩니다. 화면을 터치하는 것도 픽셀 쓰기로 간주됩니다.

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

peek2

`peek2(addr)`

주소에 있는 부호 없는 16비트 값, 리틀 엔디안.

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

poke2

`poke2(addr, val)`

부호 없는 16비트 값을 리틀 엔디안 방식으로 기록합니다.

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

peek4

`peek4(addr)`

부호 없는 32비트 값, 리틀 엔디안.

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

poke4

`poke4(addr, val)`

부호 없는 32비트 값을 리틀 엔디안 방식으로 기록합니다.

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

memcpy

`memcpy(dst, src, len)`

len 바이트만큼의 데이터를 동일한 64KB 창 안에 복사합니다.

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

memset

`memset(dest, val, len)`

dest부터 len 바이트만큼을 val 값으로 채웁니다.

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

수학

abs

`abs(x)`

절댓값.

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

flr

`flr(x)`

마이너스 무한대 방향으로 반올림합니다.

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

ceil

`ceil(x)`

무한대를 향해 반올림합니다.

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

max

`max(a, b)`

두 값 중 더 큰 값.

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

min

`min(a, b)`

두 값 중 더 작은 값.

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

mid

`mid(a, b, c)`

세 값 중 중간값입니다. 고정 장치로 유용합니다.

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

sgn

`sgn(x)`

x의 부호: -1, 0 또는 1.

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

sqrt

`sqrt(x)`

제곱근. 음의 인수는 숫자가 아닙니다.

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

sin

`sin(x)`

사인. x는 0, 0.25, 0.5, 0.75 순으로 원을 한 바퀴 도는 값입니다. 라디안이 아닙니다.

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

cos

`cos(x)`

코사인도 마찬가지로, 사인과 같은 관례를 따릅니다.

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

atan2

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

벡터 dx, dy의 각도는 0에서 1까지 변합니다.

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

rnd

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

[0, x) 범위의 난수입니다. 기본값은 x가 1이므로 rnd()는 0..1입니다. 주사위의 경우 flr(rnd(6))+1입니다.

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

srand

```
srand(seed)
```

난수 생성기의 초기값을 설정하여 이후의 rnd 시퀀스가 반복될 수 있도록 합니다.

```
srand(1)
```

band

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

비트 단위 AND 연산. 또는 a & b로도 표기합니다.

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

bor

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

비트 단위 OR 연산. 또는 | b로도 표기합니다.

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

bxor

`bxor(a, b)`

비트 단위 XOR. 또는 ^^ b라고도 씁니다.

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

bnot

`bnot(a)`

비트 단위 NOT. ~a로도 표기됩니다.

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

shl

`shl(a, n)`

왼쪽으로 이동. << n으로도 표기됩니다.

shr

`shr(a, n)`

산술 시프트 오른쪽. >> n으로도 표기합니다.

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

lshr

`lshr(a, n)`

논리적 오른쪽 이동. >>> n으로도 표기됩니다.

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

rotl

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

비트를 왼쪽으로 회전합니다. <<> n으로도 표기합니다.

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

rotr

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

비트를 오른쪽으로 회전합니다. >>< n으로도 표기합니다.

표와 값

add

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

테이블 t에 v를 추가합니다. i가 주어지면 해당 1부터 시작하는 인덱스에 삽입합니다.

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

del

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

v와 같은 첫 번째 항목을 제거합니다.

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

deli

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

인덱스 i를 제거하고 해당 값을 반환합니다. i가 없으면 마지막 항목을 제거합니다.

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

all

```
all(t)
```

배열 부분을 순서대로 순회하는 이터레이터입니다. 구멍은 1부터 시작하는 배열이 끝나는 지점에서 순회를 멈춥니다.

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

foreach

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

각 배열 요소에 대해 fn(v)를 호출합니다.

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

pairs

```
pairs(t)
```

명명된 필드를 포함하여 모든 키와 값을 순회하는 이터레이터입니다.

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

ipairs

```
ipairs(t)
```

1부터 시작하는 인덱스를 사용하여 배열 부분을 순회하는 이터레이터입니다.

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

next

`next(t, [k])`

k 다음 키와 값을 반환합니다. next(t)는 시작부터 시작합니다. 쌍으로 사용됩니다.

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

count

`count(t, [v])`

배열 부분의 길이입니다. v가 주어지면 v와 같은 항목의 개수를 셉니다.

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

pack

`pack(...)`

인수들을 새로운 표에 담습니다.

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

unpack

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

t[i]부터 t[j]까지 각각의 값을 반환합니다. 기본값은 배열 전체를 포함합니다.

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

select

`select(i, ...)`

select("#", ...)는 뒤에 오는 인수의 개수를 반환합니다. select(i, ...)는 i부터 시작하는 인수를 반환합니다.

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

assert

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

c가 false이거나 nil인 경우 오류와 함께 중지됩니다. msg는 기본 텍스트를 대체합니다.

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

pcall

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

함수 f를 호출합니다. 성공 시 true와 결과를 반환하고, 오류 시 false와 오류 텍스트를 반환합니다.

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

현악기

chr

```
chr(n)
```

바이트 코드 0~255 사이의 한 문자입니다.

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

ord

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

인덱스 i부터 j까지의 s의 바이트 코드입니다. 인덱스는 1부터 시작합니다. 기본값은 첫 번째 문자입니다.

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

sub

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

i부터 j까지의 문자열을 1부터 순차적으로 추출합니다. j를 생략하면 나머지 부분을 추출합니다.

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

split

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

s를 테이블로 분할합니다. sep의 기본값은 쉼표입니다. 숫자 sep는 청크 너비를 나타냅니다. conv는 tonum이 true인 경우 조각들을 tonum과 함께 변환합니다.

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

tonum

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

숫자를 파싱합니다. 진법은 2부터 36까지 가능합니다. 텍스트가 숫자가 아니면 nil을 반환합니다.

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

tostr

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

값을 텍스트로 변환합니다. digits는 숫자의 소수 자릿수를 설정합니다.

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

오디오

sfx

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

패턴 n을 재생하고 채널을 반환합니다. ch를 생략하거나 -1을 전달하면 첫 번째 사용 가능한 채널이 선택됩니다. ch -2는 모든 채널에서 패턴 n 재생을 중지합니다. n -1은 채널 하나를 중지하고, ch를 생략하면 모든 채널에서 중지합니다. n -2는 반복되는 패턴을 계속 재생하지 않고 종료합니다. 인자 없이 sfx()를 호출하면 -1이 반환됩니다.

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

sfxget

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

패턴 단계 하나를 나타내는 테이블을 반환합니다. [1] 피치, [2] 파형, [3] 볼륨, [4] 효과. 피치는 0..63(C-0 ~ D#5)입니다. 파형은 0..15입니다. 볼륨은 0..7이며, 0은 무음입니다. 효과는 0..7입니다.

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

sfxset

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

패턴 단계 하나를 작성합니다. 여섯 개의 인수는 모두 필수입니다.

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

sfxhdr

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

패턴 헤더. sfxhdr(n)은 헤더 테이블을 반환합니다. sfxhdr(n, key, val)은 speed, loop_in, loop_out, level, noise, crush, dtune, rverb, dampn, swing, steps 필드 중 하나를 설정합니다. 0부터 10까지의 숫자는 해당 순서대로 동일한 필드를 선택합니다. speed는 TIC(단계당 틱 수)입니다. 값이 높을수록 속도가 느려집니다. 15는 분당 120비트입니다. 16비트는 1부터 255까지의 값을 허용합니다. 좁은 범위의 신디사이저는 해당 TIC 목록에 맞춰 조정됩니다.

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

sfxstat

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

패턴 ID는 채널 ch 또는 -1에서 재생됩니다. 기본 채널은 0입니다.

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

note

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

패턴 없이 한 음을 연주합니다. 피치 0..63, 파형 0..15, 볼륨 0..7, 효과 0..7. ticks는 지속 시간을 나타내며, TIC와 동일한 120Hz 틱으로 표시됩니다. ticks를 생략하면 기본값인 15초 동안 지속됩니다. 효과가 해당 시간 동안 지속된 후 채널이 조용해집니다.

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

music

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

곡 n을 처음부터 재생합니다. music(-1)을 호출하면 재생이 중지됩니다. 이미 재생 중인 곡의 ID로 music()을 호출하면 해당 곡은 계속 실행됩니다. music()은 현재 재생 중인 곡의 ID 또는 -1을 반환합니다. fade는 밀리초 단위입니다. n이 -1이면 곡이 서서히 시작되고, -1이면 서서히 끝납니다. mask 비트는 곡에 사용되는 채널을 지정합니다. 0은 8개 채널 모두를 의미합니다. 나머지 채널은 효과음(SFX)에 사용됩니다.

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

musicstat

```
musicstat()
```

재생 중인 곡의 테이블입니다. song과 row는 각각 곡 ID와 행 번호이며, 아무것도 재생되지 않을 때는 -1입니다. step과 pos는 8자리 숫자로 구성됩니다. step[1]은 채널 0, step[8]은 채널 7입니다. step은 음표 인덱스이며, 해당 채널이 무음일 때는 -1입니다. pos는 해당 음표가 진행된 위치(0~1)를 나타냅니다. tick은 해당 행이 이미 재생된 틱 수입니다. played는 music() 함수 호출 이후 재생이 시작된 행의 총 개수(현재 재생 중인 행 포함)입니다.

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

songget

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

패턴 ID가 노래 셀에 저장됩니다. ch는 0~7입니다. 255는 셀이 비어 있음을 의미합니다. 이것은 실제 호출 내용이며, 코드 힌트 텍스트에는 여전히 이전 인자 순서가 표시됩니다.

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

songset

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

패턴 ID를 노래 셀에 기록합니다. 255를 사용하면 셀이 초기화됩니다.

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

stat

```
stat(n)
```

오디오 전용입니다. 16~23은 채널 n-16의 효과음 ID입니다. 24~31은 채널 n-24의 음악 패턴입니다. 다른 n 값은 0을 반환합니다. 프레임 타이밍 및 RAM 사용량은 성능 및 정보 제공용이며 통계용이 아닙니다.

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

데이터 테이블

dget

`dget(t, row, col)`

DATA 패널에서 데이터 테이블 t의 셀을 읽습니다. 행과 열은 0부터 시작합니다.

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

dset

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

셀에 내용을 씁니다. 텍스트 열에는 문자열을 사용하세요.

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

dstr

`dstr(t, row, col)`

셀은 텍스트 형식입니다. 숫자는 변환됩니다.

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

drows

`drows(t)`

테이블 t에는 몇 개의 행이 있습니까?

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

dcols

`dcols(t)`

테이블 t에는 열이 몇 개 있습니까?

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

저장 및 업적

save

`save(k, v)`

저장 파일의 키 k 아래에 v를 저장합니다. v는 숫자, 문자열 또는 이들의 표일 수 있습니다. 파일은 압축됩니다.

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

load

`load(k)`

키 k를 읽습니다. 키가 없으면 nil을 반환합니다.

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

delsave

`delsave(k)`

저장 파일에서 키 k를 삭제합니다.

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

saveslot

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

제작자 폴더 내의 저장 파일을 선택합니다. 파일 이름이 지정되지 않은 경우 게임 제목이 해당 슬롯에 저장됩니다. 해당 파일이 이미 존재하는 경우 1을 반환합니다.

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

achieve

```
achieve(i)
```

업적 i를 0에서 15까지 잠금 해제합니다. 로컬에 저장됩니다. Steam 빌드에서도 ACH_i가 설정됩니다. 카탈로그 세션은 이를 플레이어 계정에 보고합니다.

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

achieved

```
achieved(i)
```

1. 업적 i가 이미 잠금 해제된 경우.

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

achdef

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

업적 달성 알림 메시지는 화면 하단에서 15초 동안 표시됩니다. spr은 선택 사항인 8x8 타일이며, tex는 해당 시트입니다.

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

achclear

```
achclear(i)
```

로컬 파일에서 업적 i를 삭제합니다. 이미 설정된 Steam ACH_i는 그대로 유지됩니다.

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

회로망

whoami

```
whoami()
```

세션이 활성화된 동안에는 플레이어 ID가 0~15 사이입니다. 오프라인 상태에서는 -1입니다.

```
id = whoami()
```

nhost

```
nhost()
```

이 기기가 호스트이면 1, 그렇지 않으면 0입니다.

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

nplayers

```
nplayers()
```

0~15에 존재하는 활성 ID는 몇 개입니까?

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

nlisten

`nlisten()`

UDP 호스팅을 시작하고 LAN 주소를 ip:port 형식으로 반환합니다.

```
addr = nlisten()
```

njoin

`njoin(addr)`

IP 주소 또는 IP 주소:포트로 접속합니다.

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

nleave

`nleave()`

세션을 나갑니다. 다른 사용자는 이유가 0인 _mp_leave를 보게 됩니다.

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

nsend

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

손실이 있는 메시지를 전송합니다. 이름은 이벤트입니다. 최대 4개의 추가 값이 뒤따릅니다. p는 0에서 15 사이의 값으로 해당 피어에게 귓속말을 보냅니다. p를 생략하면 모든 사람에게 전송됩니다.

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

ncall

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

nsend와 동일한 형식이지만, 수신 확인이 있을 때까지 메시지 전송을 재시도합니다. 연속으로 세 번 실패하면 세션이 종료됩니다.

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

nlan

```
nlan()
```

nlisten 실행 후 현재 LAN IP:포트입니다.

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

nstun

```
nstun()
```

STUN에서 제공하는 공용 IP 주소와 포트 번호입니다. 다른 네트워크의 게스트가 해당 네트워크에 접속을 시도할 수 있습니다.

```
pub = nstun()
```

setturn

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

직접 경로가 실패할 경우 TURN이 대체 경로로 사용됩니다. setturn() 함수에 인수를 비워두면 TURN이 비활성화됩니다. 주소는 호스트:포트 형식입니다.

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

nlist

`nlist(game)`

해당 게임 이름의 진행 중인 세션 목록입니다. 각 항목에는 주소, 참가자 수 및 이름이 포함되어 있습니다.

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

nannounce

`nannounce(game)`

하트비트 기능이 활성화되면 호스트가 게임 목록에 표시됩니다. 호스팅 중에 타이머를 설정하여 하트비트 기능을 사용하세요.

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

nkb

`nkb([on])`

주소 필드. nkb(1)은 필드를 열고, nkb(0)은 필드를 숨기며, nkb()는 필드가 열려 있는지 여부를 알려줍니다. 위쪽 및 아래쪽 키는 문자(0~9, 점 또는 콜론)를 변경합니다. A는 입력을, X는 삭제를 합니다. 키보드로는 필드에 입력할 수 없습니다.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

주소 필드에 텍스트가 입력되면 위아래 문자가 바뀝니다. nkbstr("")는 텍스트를 지웁니다.

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

_mp

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

이 함수는 메시지가 도착했을 때 실행됩니다. `from`은 발신자 ID이고, `name`은 이벤트 이름입니다. 그 뒤에는 최대 네 개의 값이 올 수 있습니다.

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

_mp_join

`_mp_join(p)`

이 함수를 작성하세요. 상대방이 ID p를 받았습니다. 이 함수는 당신에게도 실행됩니다. whoami()가 바로 그 ID입니다.

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

_mp_leave

`_mp_leave(p, why)`

이 함수를 작성하세요. 피어 p가 나갔습니다. 0은 일반적인 나감이며, 1은 연결 끊김을 의미합니다.

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

_mp_update

`_mp_update()`

이 함수를 작성하세요. 이 함수는 15개의 _update 프레임마다 실행되며, 약 4Hz의 빈도로 실행됩니다. 위치 정보를 전송하기에 좋은 위치입니다.

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

_mp_fail

`_mp_fail()`

이 함수를 작성했습니다. 세션이 종료되었습니다. 호스트가 사라졌거나 ncall이 세 번 실패했습니다.

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

코루틴

cocreate

`cocreate(f)`

함수 f로부터 코루틴을 생성합니다. 이 코루틴은 coresume이 호출될 때까지 실행되지 않습니다.

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

coresume

`coresume(c, ...)`

코루틴 c가 실행되어 결과를 반환할 때까지 계속 실행합니다. 성공 시: true와 반환된 값을 반환합니다. 오류 시: false와 오류 메시지를 반환합니다.

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

costatus

```
costatus(c)
```

"정지됨", "진행 중", 또는 "중단됨".

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

yield

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

coresume를 호출한 사용자에게 제어권을 되돌려줍니다. 값은 coresume의 결과가 됩니다.

```
yield(1)
```

측정하다

info

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

세션이 활성화되면 _draw 함수에서 실시간 HUD를 그립니다. 출력되는 정보는 fps(1초 평균), upd, drw, px, max, ram, peak, ramc, net입니다. info() 함수는 출력 커서를 사용합니다. info(col)은 색상만 설정하고, info(x, y) 또는 info(x, y, col)은 위치를 지정합니다. 숫자는 작은 화면에 맞게 B, k 또는 m 단위로 표시됩니다. 함수 호출 자체는 픽셀 수 계산에 포함되지 않습니다.

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

showstats

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

정보와 동일한 HUD이며, _draw 이후 0, 0에 자동으로 그려지고 카메라와 클립을 무시합니다. showstats(0)은 숨깁니다. showstats(1) 또는 showstats()는 표시합니다.

```
showstats(1)
```

perf

`perf(n)`

HUD 숫자는 하나뿐이므로 직접 그릴 수 있습니다. 0fps, 1upd, 2drw, 3px, 4max(초당 가장 부하가 큰 프레임), 5ram, 6ramc(RAM + 게임 카트리지 용량), 7 게임 카트리지 시작 이후 최대 RAM 사용량.

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