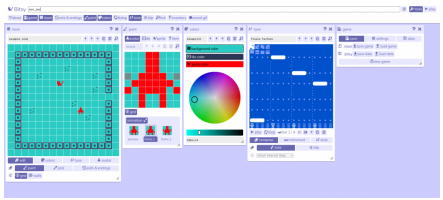
jeu-video, narration, game-jam, pixel-art, em

Bitsy



Bitsy est un **moteur de création de jeux vidéo en 2D** simple et accessible, conçu pour créer des jeux d'aventure ou des récits interactifs de manière ludique et visuelle. L'application permet de réaliser des graphismes en pixel art, des musiques et effets sonores, d'ajouter des dialogues et des interactions sans nécessiter de compétences en programmation. Bitsy permet également de définir des règles de gameplay simples, comme des déplacements, des actions ou des changements de scène, tout en offrant un export facile pour partager les jeux créés en ligne.

- Utilisable en ligne : https://www.bitsy.org/
- Faire des jeux : https://make.bitsy.org/
- documentation : https://make.bitsy.org/docs/index.html
- code source : https://codeberg.org/adamledoux/bitsy

De nombreux jeux ont été réalisé, on peut également y jouer grâce à cet émulateur : https://ledoux.itch.io/bitsybox (fonctionne sur rpi4)
~~Attention, on peut jouer uniquement aux jeux créés avec la version 7.12 max au 15 nov 2024, pas à ceux de la version 8 actuellement en ligne.~~
Maintenant, c'est possible de jouer aux jeux de la version 8 avec bitsybox : https://ledoux.itch.io/bitsybox/devlog/840940/bitsybox-supports-bitsy-8

Adam Ledoux, à l'origine de Bitsy partage ses réflexions sur la conservation des jeux Bitsy : https://ledoux.itch.io/bitsy/devlog/359498/some-thoughts-on-bitsybox-and-game-preservation

6 boutons sont nécessaires pour Bitsy :

- N'importe quelle touche du clavier : clic bouton
- flèches directionnelles pour déplacer l'avatar
- CTRL + R : retour au démarrage du jeu

Bitsybox

- ALT + Entrée : plein écran
- ALT + F4 : quitter
- CTRL + R ou ESC : retour au menu
- flèches directionnelles pour déplacer l'avatar
- Espace ou Entrée : ok / suite

Pour ajouter un jeu, l'ajouter au dossier «games» avec l'extension .bitsy, les jeux sont exportables depuis l'interface web (game data tool → download data). Si ce dossier contient un seul jeu, il sera démarré automatiquement, sinon bitsybox démarrera sur le menu.

Utiliser Bitsy sans internet

En l'installant avec le navigateur : https://ledoux.itch.io/bitsy/devlog/274859/v76-new-mobile-layout

Ou en utilisant bitsy-savior, sous forme d'application : https://aloelazoe.itch.io/bitsy-savior

Installation de bitsy-savior

téléchargement : https://aloelazoe.itch.io/bitsy-savior / https://github.com/aloelazoe
C'est une appli. electron, fourni avec Bitsy 7.2, mais tenter de le démarrer déclenche un message d'erreur «*an error occured trying to load the editor*».
C'est une erreur déjà documentée (https://github.com/aloelazoe/bitsy-savior/issues/21) mais pas trouvé de solution... Par contre, on peut ouvrir le fichier `/bitsy-savior-10.1.5/resources/app/src/bitsy/editor/index.html` et tout semble fonctionner correctement

Des outils utiles pour Bitsy

- Pixsy : pixeliser des images et les importer dans Bitsy https://ruin.itch.io/pixsy
- Bitsy merge : https://seansleblanc.itch.io/bitsy-merge

Ainsi que d'autres versions de bitsy (plus compliquées à utiliser)

- Bitsycolor : https://aurysystem.itch.io/multicolorbisty
- Bitsy 3D : https://aloelazoe.itch.io/bitsy-3d
- Bipsi (variante) https://kool.tools/bipsi/

Extraire les parties graphiques

Le script suivant permet d'extraire la partie graphique d'un fichier bitsy (au format .html ou .bitsy) et de la mettre sous forme de 2 spritesheets : s'il n'existe qu'une frame d'animation elle est répétée sur les deux feuilles, s'il existe deux frames elles sont réparties une par feuille. Puis ces deux feuilles sont assemblées pour constituer un gif animé.

python3 bitsy_extract_sprites_anim.py (cliquer pour afficher le code)

python3 bitsy_extract_sprites_anim.py

```
#!/usr/bin/env python3
"""Extrait les graphismes d'un jeu Bitsy : 2 spritesheets PNG + GIF animés (stdlib uniquement).
20261026 / réalisé par un LLM

arguments :

-o / --output (par défaut : spritesheet.png) : nom de base : produit <nom>_1.png, <nom>_2.png, <nom>.gif, <nom>.json
-c / --columns (par défaut : 16) : colonnes
-s / --scale (par défaut : 1) : facteur d'agrandissement
-p / --palette (par défaut : 1) : id de palette
-d / --delay (par défaut : 400) : durée d'une frame en ms
--opaque : fond de palette aussi pour sprites et objets (sinon transparent)
--no-individual : ne pas générer un GIF par graphique

exemples :

python3 bitsy_extract_sprites_anim.py mon_jeu.html -o sheet.png -s 8 -c 8
# -> sheet_1.png, sheet_2.png, sheet.gif, sheet.json, sheet_gifs/*.gif

python3 bitsy_extract_sprites_anim.py mon_jeu.html -d 250 --no-individual

"""

import argparse
import json
import re
import struct
import sys
import zlib
from pathlib import Path

PIXEL_ROW = re.compile(r"[01]+")
RGB = re.compile(r"\s*(\d+)\s*,\s*(\d+)\s*,\s*(\d+)\s*$")
HEADER = re.compile(r"(PAL|TIL|SPR|ITM)\s+(\S+)\s*$")

# Index de couleur par défaut dans une palette Bitsy : 0 = fond, 1 = tuiles, 2 = sprites/objets
DEFAULT_COL = {"TIL": 1, "SPR": 2, "ITM": 2}
KIND_LABEL = {"TIL": "tile", "SPR": "sprite", "ITM": "item"}
KIND_ORDER = {"TIL": 0, "SPR": 1, "ITM": 2}

BITSY_FRAME_MS = 400


# ---------------------------------------------------------------------------- #
# Lecture des données Bitsy
# ---------------------------------------------------------------------------- #
```

```
def extract_game_data(text: str) -> str:
    """Retourne les données Bitsy, qu'il s'agisse d'un export HTML ou d'un fichier brut."""
    m = re.search(
        r'<script[^>]*id=["\']exportedGameData["\'][^>]*>(.*?)</script>', text, re.S
    )
    return m.group(1) if m else text


def parse_bitsy(data: str):
    """Retourne (palettes, graphiques)."""
    palettes = {}
    graphics = []

    for block in re.split(r"\n\s*\n", data.replace("\r\n", "\n").replace("\r", "\n")):
        lines = [l.strip() for l in block.strip().split("\n")]
        if not lines:
            continue
        m = HEADER.match(lines[0])
        if not m:
            continue
        kind, gid = m.groups()

        if kind == "PAL":
            palettes[gid] = [
                tuple(int(v) for v in c.groups())
                for c in (RGB.match(l) for l in lines[1:])
                if c
            ]
            continue

        frames = [[]]
        name = None
        col = DEFAULT_COL[kind]
        for l in lines[1:]:
            if PIXEL_ROW.fullmatch(l):
                frames[-1].append(l)
            elif l == ">":
                frames.append([])
            elif l.startswith("NAME "):
                name = l[5:].strip()
            elif l.startswith("COL "):
                try:
                    col = int(l[4:].strip())
                except ValueError:
                    pass
        frames = [f for f in frames if f]
        if frames:
            graphics.append(
                {"kind": kind, "id": gid, "name": name, "col": col, "frames": frames}
            )

    return palettes, graphics


# ---------------------------------------------------------------------------- #
# Rendu en pixels indexés
# ---------------------------------------------------------------------------- #

def make_color_table(palette):
    """Table de couleurs : couleurs de la palette + une entrée transparente.

    Retourne (table_complète_pour_GIF, bits, index_transparent, couleurs)."""
    colors = [
        palette[i] if i < len(palette) else (255, 255, 255)
        for i in range(max(3, len(palette)))
    ]
    transparent = len(colors)
    bits = max(2, transparent.bit_length())  # transparent = dernier index utilisé
    table = colors + [(0, 0, 0)] * ((1 << bits) - len(colors))
    return table, bits, transparent, colors


def render_sheet(graphics, frame_idx, columns, rows, cell_w, cell_h, scale,
                 opaque, transparent, ncolors):
    """Dessine une feuille en indices de palette (1 octet par pixel).

    frame_idx vaut 0 ou 1 ; un graphique à une seule frame est répété."""
    W, H = columns * cell_w * scale, rows * cell_h * scale
    buf = bytearray([transparent]) * (W * H)

    for i, g in enumerate(graphics):
        frame = g["frames"][frame_idx] if frame_idx < len(g["frames"]) else g["frames"][0]
        cx, cy = (i % columns) * cell_w, (i // columns) * cell_h
        fg = g["col"] if g["col"] < ncolors else 1
        bg = 0 if (g["kind"] == "TIL" or opaque) else transparent  # tuiles toujours opaques
        for y, row in enumerate(frame):
            line = b"".join(bytes([fg if px == "1" else bg]) * scale for px in row)
            for dy in range(scale):
                off = ((cy + y) * scale + dy) * W + cx * scale
                buf[off:off + len(line)] = line
    return W, H, buf


def crop(buf, W, x, y, w, h):
```

```
    return b"".join(buf[(y + r) * W + x:(y + r) * W + x + w] for r in range(h))


# ---------------------------------------------------------------------------- #
# Encodeur PNG
# ---------------------------------------------------------------------------- #

def write_png(path, width, height, indices, colors, transparent):
    # Conversion indices -> RGBA par translate() (rapide)
    def lut(channel):
        t = bytearray(256)
        for i, c in enumerate(colors):
            t[i] = c[channel]
        return bytes(t)

    alpha = bytearray(b"\xff" * 256)
    alpha[transparent] = 0

    raw_rgba = bytearray(width * height * 4)
    raw_rgba[0::4] = bytes(indices).translate(lut(0))
    raw_rgba[1::4] = bytes(indices).translate(lut(1))
    raw_rgba[2::4] = bytes(indices).translate(lut(2))
    raw_rgba[3::4] = bytes(indices).translate(bytes(alpha))

    def chunk(tag, payload):
        body = tag + payload
        return struct.pack(">I", len(payload)) + body + struct.pack(">I", zlib.crc32(body))

    stride = width * 4
    raw = b"".join(b"\x00" + bytes(raw_rgba[y * stride:(y + 1) * stride]) for y in range(height))
    png = (
        b"\x89PNG\r\n\x1a\n"
        + chunk(b"IHDR", struct.pack(">IIBBBBB", width, height, 8, 6, 0, 0, 0))
        + chunk(b"IDAT", zlib.compress(raw, 9))
        + chunk(b"IEND", b"")
    )
    Path(path).write_bytes(png)


# ---------------------------------------------------------------------------- #
# Encodeur GIF (LZW)
# ---------------------------------------------------------------------------- #

def lzw_encode(data, min_size):
    """Compression LZW au format GIF (codes de largeur variable, bits LSB d'abord)."""
    clear = 1 << min_size
    eoi = clear + 1
    out = bytearray()
    bitbuf = 0
    bitcnt = 0
    size = min_size + 1

    def emit(code):
        nonlocal bitbuf, bitcnt
        bitbuf |= code << bitcnt
        bitcnt += size
        while bitcnt >= 8:
            out.append(bitbuf & 0xFF)
            bitbuf >>= 8
            bitcnt -= 8

    table = {}
    next_code = eoi + 1
    emit(clear)
    prefix = data[0]

    for px in data[1:]:
        key = (prefix, px)
        code = table.get(key)
        if code is not None:
            prefix = code
            continue
        emit(prefix)
        if next_code < 4096:
            if next_code >= (1 << size) and size < 12:
                size += 1
            table[key] = next_code
            next_code += 1
        else:  # table pleine : on la réinitialise
            emit(clear)
            table.clear()
            next_code = eoi + 1
            size = min_size + 1
        prefix = px

    emit(prefix)
    if next_code >= (1 << size) and size < 12:
        size += 1
    emit(eoi)
    if bitcnt:
        out.append(bitbuf & 0xFF)
    return bytes(out)
```

```python
def sub_blocks(data):
    out = bytearray()
    for i in range(0, len(data), 255):
        chunk = data[i:i + 255]
        out.append(len(chunk))
        out += chunk
    out.append(0)
    return bytes(out)


def write_gif(path, w, h, frames, table, bits, transparent, delay_ms):
    """Écrit un GIF (animé en boucle si plusieurs frames) à partir de frames indexées."""
    delay_cs = max(1, round(delay_ms / 10))
    out = bytearray(b"GIF89a")
    out += struct.pack("<HHBBB", w, h, 0x80 | ((bits - 1) << 4) | (bits - 1), 0, 0)
    for r, g, b in table:
        out += bytes((r, g, b))
    if len(frames) > 1:
        out += b"\x21\xff\x0bNETSCAPE2.0\x03\x01\x00\x00\x00"  # boucle infinie
    for frame in frames:
        # disposal 2 = on efface avant la frame suivante (indispensable avec la transparence)
        out += b"\x21\xf9\x04" + bytes([(2 << 2) | 1]) + struct.pack("<H", delay_cs)
        out += bytes([transparent]) + b"\x00"
        out += b"\x2c" + struct.pack("<HHHHB", 0, 0, w, h, 0)
        out += bytes([bits]) + sub_blocks(lzw_encode(bytes(frame), bits))
    out += b"\x3b"
    Path(path).write_bytes(bytes(out))


# ---------------------------------------------------------------------------- #
# Programme principal
# ---------------------------------------------------------------------------- #

def safe(s):
    return re.sub(r"[^\w\-]+", "_", s).strip("_") or "x"


def main():
    ap = argparse.ArgumentParser(description="Spritesheets et GIF animés à partir d'un jeu Bitsy")
    ap.add_argument("source", help="fichier .bitsy / .txt / .html exporté")
    ap.add_argument("-o", "--output", default="spritesheet.png",
                    help="nom de base : produit <nom>_1.png, <nom>_2.png, <nom>.gif, <nom>.json")
    ap.add_argument("-c", "--columns", type=int, default=16, help="colonnes (défaut 16)")
    ap.add_argument("-s", "--scale", type=int, default=1, help="facteur d'agrandissement")
    ap.add_argument("-p", "--palette", default=None, help="id de palette (défaut : première)")
    ap.add_argument("-d", "--delay", type=int, default=BITSY_FRAME_MS,
                    help=f"durée d'une frame en ms (défaut {BITSY_FRAME_MS}, cadence de Bitsy)")
    ap.add_argument("--opaque", action="store_true",
                    help="fond de palette aussi pour sprites et objets (sinon transparent)")
    ap.add_argument("--no-individual", action="store_true",
                    help="ne pas générer un GIF par graphique")
    args = ap.parse_args()

    text = Path(args.source).read_text(encoding="utf-8", errors="replace")
    palettes, graphics = parse_bitsy(extract_game_data(text))

    if not graphics:
        sys.exit("Aucun graphique (TIL/SPR/ITM) trouvé.")

    if palettes:
        pid = args.palette if args.palette is not None else next(iter(palettes))
        if pid not in palettes:
            sys.exit(f"Palette '{pid}' introuvable. Disponibles : {', '.join(palettes)}")
        palette = palettes[pid]
    else:
        palette = [(0, 0, 0), (255, 255, 255), (255, 255, 255)]

    table, bits, transparent, colors = make_color_table(palette)

    graphics.sort(key=lambda g: (KIND_ORDER[g["kind"]], g["id"]))
    cell_w = max(len(r) for g in graphics for f in g["frames"] for r in f)
    cell_h = max(len(f) for g in graphics for f in g["frames"])
    columns = max(1, min(args.columns, len(graphics)))
    rows = -(-len(graphics) // columns)
    s = args.scale

    # Deux feuilles indexées (même disposition)
    sheets = []
    for n in (0, 1):
        W, H, buf = render_sheet(graphics, n, columns, rows, cell_w, cell_h, s,
                                 args.opaque, transparent, len(colors))
        sheets.append(buf)

    out = Path(args.output)
    suffix = out.suffix or ".png"

    # PNG
    png_paths = []
    for n, buf in enumerate(sheets):
        p = out.with_name(f"{out.stem}_{n + 1}{suffix}")
        write_png(p, W, H, buf, colors, transparent)
        png_paths.append(p)

    # GIF de la feuille complète
```

```
    sheet_gif = out.with_suffix(".gif")
    write_gif(sheet_gif, W, H, sheets, table, bits, transparent, args.delay)

    # GIF individuels
    gif_dir = out.with_name(out.stem + "_gifs")
    if not args.no_individual:
        gif_dir.mkdir(parents=True, exist_ok=True)

    index = []
    for i, g in enumerate(graphics):
        x, y = (i % columns) * cell_w * s, (i // columns) * cell_h * s
        w, h = cell_w * s, cell_h * s
        animated = len(g["frames"]) > 1
        entry = {
            "type": KIND_LABEL[g["kind"]],
            "id": g["id"],
            "name": g["name"],
            "animated": animated,
            "x": x, "y": y, "w": w, "h": h,
        }
        if not args.no_individual:
            frames = [crop(sheets[0], W, x, y, w, h)]
            if animated:
                frames.append(crop(sheets[1], W, x, y, w, h))
            label = f"{KIND_LABEL[g['kind']]}_{safe(g['id'])}"
            if g["name"]:
                label += f"_{safe(g['name'])}"
            gif_path = gif_dir / f"{label}.gif"
            write_gif(gif_path, w, h, frames, table, bits, transparent, args.delay)
            entry["gif"] = gif_path.name
        index.append(entry)

    index_path = out.with_suffix(".json")
    index_path.write_text(json.dumps(index, indent=2, ensure_ascii=False), encoding="utf-8")

    print(f"{len(graphics)} graphiques ({W}x{H}) -> {png_paths[0]}, {png_paths[1]}, "
          f"{sheet_gif}" + (f", {gif_dir}/" if not args.no_individual else "")
          + f" + {index_path}")


if __name__ == "__main__":
    main()
```

Jeux

- Super Bitsy Land (demake) : https://pocketfruit.itch.io/super-bitsy-land
- Bitsy-100, gamejam (oct 2026/janvier 2027) : https://itch.io/jam/bitsy-100 sur la page les liens vers les gamejams Bitsy précédentes)
- La page itch d'Adam Ledoux : https://ledoux.itch.io/
- Laundry (par Juno Morrow) : https://junomorrow.itch.io/laundry
- Queer Picnic (de Rae White) : https://raewhite.itch.io/queer-picnic

Ressources

- Forum : https://ledoux.itch.io/bitsy/community
- Mastodon : https://digipres.club/@bitsy

- **Bitsy Hacks** : https://github.com/seleb/bitsy-hacks
- **Borsky** https://ayolland.itch.io/borksy
- https://nowplaythis.net/game/faces-of-bitsy/
- https://brandonmakesthings.itch.io/bitsy-cheatsheet
- https://ripplewrites.neocities.org/bitsyguide
- https://gameswemake.com/?page_id=93
- https://itch.io/c/2840780/bitsy-resources
- https://bitsy.org/friends.html
- https://bitsy.fandom.com/wiki/Bitsy_Wiki

Tilesets

- forest zone (Adam Ledoux) : https://ledoux.itch.io/forest-zone-bitsy-tile-pack
- terrarium (Adam Ledoux) : https://ledoux.itch.io/terrarium-bitsy-tile-pack
- safe as houses (Anna Anthropy) : https://w.itch.io/safe-as-houses
- dungeon (enui) : https://enui.itch.io/bitsy-dungeon-tiles
- nothin'fancy (Anna Anthropy) : https://w.itch.io/nothin-fancy

Article extrait de : https://lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse : **https://lesporteslogiques.net/wiki/ressource/logiciel/bitsy/start?rev=1793029754**
Article mis à jour: **2026/10/26 16:49**