Not Tetris

Auteur	Catégorie	Niveau
Cécile	Reverse	Facile

Énoncé :

Intro

Ce jeu cache un secret, il faut atteindre 10 000 lignes pour le découvrir, y arriveras tu ?

Pièce Jointe

Un fichier exécutable tetris.

Préparation du challenge

Objectif

Trouver ce que cache ce que cache ce jeu, on peut essayer d'atteindre les 10 000 points mais cela risque d'être très long alors le mieux est d'essayer de faire du reverse engineering sur ce fichier.

Flag

NHM2I{Sc0r3-m4x-Br4v0!}

Démarche

On code un tetris à partir d'un modèle en python, on adapte le code pour que le jeu s'arrête à un certain niveau de score et affiche le flag. On joue également sur les couleurs pour qu'il soit plus lisible.

```
#!/usr/bin/python
# -*-coding:Utf-8 -*

import pygame
import random

# Création d'une bibliothèque de couleurs utilisées pour colorer les pièces, ici on utilise des pastels

# VIVES
couleurs = [
    (230, 176, 170),
    (241, 116, 91),
    (252, 206, 182),
    (255, 218, 141),
    (234, 241, 213),
    (184, 192, 104),
    (159, 197, 170),
    (171, 217, 213),
    (169, 214, 235),
    (173, 153, 200),
]


# On créé les formes dans un tableau de 4 x 4 cases.
# On créé les tuples pour chaque forme et leurs rotations.
class Figure:
    x = 0
    y = 0

    figures = [
        # Ligne
        [[1, 5, 9, 13], [4, 5, 6, 7]],
        # Z
        [[4, 5, 9, 10], [2, 6, 5, 9]],
        [[6, 7, 9, 10], [1, 5, 6, 10]],
        # L
        [[1, 2, 5, 9], [0, 4, 5, 6], [1, 5, 9, 8], [4, 5, 6, 10]],
        [[1, 2, 6, 10], [5, 6, 7, 9], [2, 6, 10, 11], [3, 5, 6, 7]],
        # T
        [[1, 4, 5, 6], [1, 4, 5, 9], [4, 5, 6, 9], [1, 5, 6, 9]],
        # Carré
        [[1, 2, 5, 6]],
    ]

    # On initialise la classe Figure avec ces caractéristiques
    # (positions x et y, type de figure, couleur de figure, état de rotation initial) :

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.type = random.randint(0, len(self.figures) - 1)
        self.couleur = random.randint(1, len(couleurs) - 1)
        self.rotation = 0

    def image(self):
        return self.figures[self.type][self.rotation]
```

```
    def rotate(self):
        self.rotation = (self.rotation + 1) % len(self.figures[self.type])


# On créé la classe Tetris avec les caractéristiques du jeu.
class Tetris:
    level = 1
    score = 0
    state = "start"
    field = []
    height = 0
    width = 0

    # Emplacement et taille de la grille par rapport à la fenêtre ...
    x = 100
    y = 60
    zoom = 20
    figure = None

    def __init__(self, height, width):
        self.height = height
        self.width = width
        self.field = []
        self.score = 0
        self.state = "start"
        for i in range(height):
            new_line = []
            for j in range(width):
                new_line.append(0)
            self.field.append(new_line)

    def new_figure(self):
        self.figure = Figure(3, 0)

    def intersects(self):
        intersection = False
        for i in range(4):
            for j in range(4):
                if i * 4 + j in self.figure.image():
                    if i + self.figure.y > self.height - 1 or \
                            j + self.figure.x > self.width - 1 or \
                            j + self.figure.x < 0 or \
                            self.field[i + self.figure.y][j + self.figure.x] > 0:
                        intersection = True
        return intersection

    def break_lines(self):
        lines = 0
        for i in range(1, self.height):
            zeros = 0
            for j in range(self.width):
                if self.field[i][j] == 0:
                    zeros += 1
            if zeros == 0:
                lines += 1
                for i1 in range(i, 1, -1):
                    for j in range(self.width):
                        self.field[i1][j] = self.field[i1 - 1][j]
        self.score += lines ** 2

    def go_space(self):
        while not self.intersects():
            self.figure.y += 1
        self.figure.y -= 1
        self.freeze()

    def go_down(self):
        self.figure.y += 1
        if self.intersects():
            self.figure.y -= 1
            self.freeze()
# On définie le comportement du jeu.
    def freeze(self):
        for i in range(4):
            for j in range(4):
                if i * 4 + j in self.figure.image():
                    self.field[i + self.figure.y][j + self.figure.x] = self.figure.couleur
        self.break_lines()
        self.new_figure()
        if self.intersects():
            self.state = "gameover"
        if self.score >= 10000 :
            self.state = "winner"

    def go_side(self, dx):
        old_x = self.figure.x
        self.figure.x += dx
        if self.intersects():
            self.figure.x = old_x

    def rotate(self):
        old_rotation = self.figure.rotation
        self.figure.rotate()
        if self.intersects():
            self.figure.rotation = old_rotation


# On initialise le jeu avec pygame
pygame.init()

# Define some colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
```

```
GRAY = (128, 128, 128)

size = (400, 500)
screen = pygame.display.set_mode(size)

pygame.display.set_caption("Tetris")

# Loop until the user clicks the close button.
done = False
clock = pygame.time.Clock()
fps = 25
game = Tetris(20, 10)
counter = 0

pressing_down = False

    # On ajoute un compteur pour calculer l'atteinte d'un certain niveau de point (normalement il ne s'arrête que quand c'est perdu)
while not done:
    if game.figure is None:
        game.new_figure()
    counter += 1
    if counter > 100000:
        counter = 0

    if counter % (fps // game.level // 2) == 0 or pressing_down:
        if game.state == "start":
            game.go_down()

# On définie le comportement du jeu lors de l'utilisation des touches

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                game.rotate()
            if event.key == pygame.K_DOWN:
                pressing_down = True
            if event.key == pygame.K_LEFT:
                game.go_side(-1)
            if event.key == pygame.K_RIGHT:
                game.go_side(1)
            if event.key == pygame.K_SPACE:
                game.go_space()
            if event.key == pygame.K_ESCAPE:
                game.__init__(20, 10)

    if event.type == pygame.KEYUP:
            if event.key == pygame.K_DOWN:
                pressing_down = False

    screen.fill(BLACK)

    for i in range(game.height):
        for j in range(game.width):
            pygame.draw.rect(screen, GRAY, [game.x + game.zoom * j, game.y + game.zoom * i, game.zoom, game.zoom], 1)
            if game.field[i][j] > 0:
                pygame.draw.rect(screen, couleurs[game.field[i][j]],
                                 [game.x + game.zoom * j + 1, game.y + game.zoom * i + 1, game.zoom - 2, game.zoom - 1])

    if game.figure is not None:
        for i in range(4):
            for j in range(4):
                p = i * 4 + j
                if p in game.figure.image():
                    pygame.draw.rect(screen, couleurs[game.figure.couleur],
                                     [game.x + game.zoom * (j + game.figure.x) + 1,
                                      game.y + game.zoom * (i + game.figure.y) + 1,
                                      game.zoom - 2, game.zoom - 2])

    font = pygame.font.SysFont('Calibri', 35, True, False)
    font1 = pygame.font.SysFont('Calibri', 20, True, False)
    text = font.render("Score : " + str(game.score), True, WHITE)

    # On modifie le comportement du jeu pour qu'il affiche un flag lors de l'atteinte d'un certain niveau de point (normalement il ne s'arrête que
quand c'est perdu)
    text_gagne = font1.render("=> NHM2I{Sc0r3-m4x-Br4v0!}", True, (118, 215, 196))
    text_gagne1 = font.render("SURPRISE !!!", True, (230, 6, 45))
    text_game_over = font.render("GAME OVER", True, (230, 6, 45))
    text_game_over1 = font1.render("Appuyer sur ESC pour rejouer", True, (255, 255, 255))

    # blit() - blit signifie Block Transfer - et il va copier le contenu d'une Surface sur une autre Surface . 00:17 Les deux surfaces en question
sont
    # l'écran que vous avez créé et la nouvelle Surface . Donc, . blit() va prendre cette Surface rectangulaire et la placer au dessus de l'écran.
    # La méthode prend deux arguments.

    screen.blit(text, [0, 0])
    if game.state == "gameover":
        screen.blit(text_game_over, [80, 250])
        screen.blit(text_game_over1, [30, 465])
    if game.state == "winner" :
        screen.blit(text_gagne1, [80, 250])
        screen.blit(text_gagne, [30, 465])


    pygame.display.flip()
    clock.tick(fps)

pygame.quit()
```

Une fois le fichier python fonctionnel, on le transforme en fichier exécutable :

On va utiliser PyInstaller, on l'installe :

```
pip install pyinstaller
```

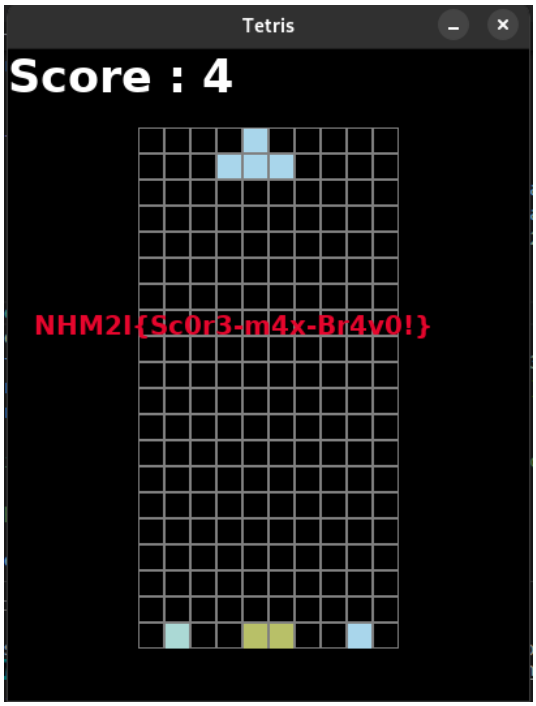
Pyinstaller est un module Python qui permet de convertir le fichier Python en un fichier exécutable. Il regroupe une application Python et toutes ses dépendances dans un seul package. Ainsi, vous pouvez convertir le fichier Python en application et l'utilisateur peut l'exécuter sans installer Python et ses packages.

On exécute la commande permettant de transformer le fichier .py en un fichier executable :

```
pyinstaller --onefile tetris
```

Lorsqu'on exécute cette commande, le processus de conversion du fichier Python en *exe* se déclenche et crée trois répertoires. Le répertoire avec le nom ***dist*** contient le fichier *exe*.

```
119
120     def freeze(self):
121         for i in range(4):
122             for j in range(4):
123                 if i * 4 + j in self.figure.image():
124                     self.field[i + self.figure.y][j + self.figure.x] = self.figure.couleur
125         self.break_lines()
126         self.new_figure()
127         if self.intersects():
128             self.state = "gameover"
129         if self.score >= 4 :
130             self.state = "winner"
```



Résolution :

Solution débutant ++ :

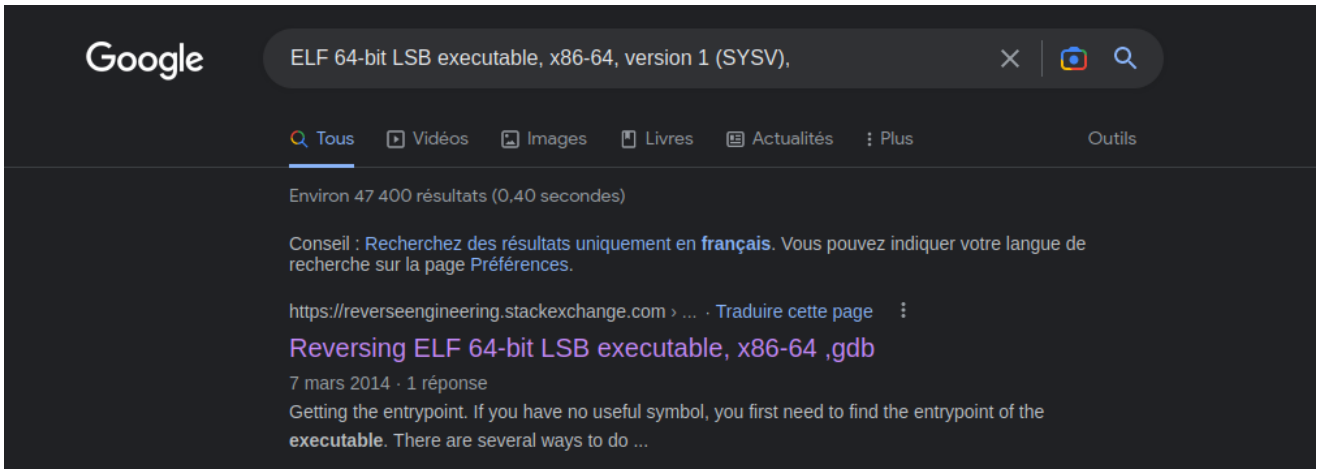
On essaye d'identifier le type de fichier :

```
file tetris
```

On obtient le résultat suivant :

```
tetris: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=8203f620bc278dd6a87dfca5cfeebceecc1998bd, for GNU/Linux 2.6.32, stripped
```

On cherche sur internet comment faire du reverse sur ce type de fichier :



On installe gdb :

```
sudo pacman -Syu gdb
```

Cela ne semble rien donner. On essaye la commande `strings` et a on obtient un peu plus d'informations. On voit qu'il y a des bibliothèques propres python comme pygame.

```
blibz.so.1
bmarkupsafe/_speedups.cpython-310-x86_64-linux-gnu.so
bnumpy/core/_multiarray_tests.cpython-310-x86_64-linux-gnu.so
bnumpy/core/_multiarray_umath.cpython-310-x86_64-linux-gnu.so
bnumpy/fft/_pocketfft_internal.cpython-310-x86_64-linux-gnu.so
bnumpy/linalg/_umath_linalg.cpython-310-x86_64-linux-gnu.so
bnumpy/linalg/lapack_lite.cpython-310-x86_64-linux-gnu.so
bnumpy/random/_bounded_integers.cpython-310-x86_64-linux-gnu.so
bnumpy/random/_common.cpython-310-x86_64-linux-gnu.so
bnumpy/random/_generator.cpython-310-x86_64-linux-gnu.so
bnumpy/random/_mt19937.cpython-310-x86_64-linux-gnu.so
bnumpy/random/_pcg64.cpython-310-x86_64-linux-gnu.so
bnumpy/random/_philox.cpython-310-x86_64-linux-gnu.so
bnumpy/random/_sfc64.cpython-310-x86_64-linux-gnu.so
bnumpy/random/bit_generator.cpython-310-x86_64-linux-gnu.so
bnumpy/random/mtrand.cpython-310-x86_64-linux-gnu.so
bpygame/_freetype.cpython-310-x86_64-linux-gnu.so
bpygame/base.cpython-310-x86_64-linux-gnu.so
bpygame/bufferproxy.cpython-310-x86_64-linux-gnu.so
bpygame/color.cpython-310-x86_64-linux-gnu.so
bpygame/constants.cpython-310-x86_64-linux-gnu.so
```

On cherche donc comment faire du reverse sur des fichiers python.

On créé un dump du fichier ELF tetris sous le nom pydata.dump :

```
$ objcopy --dump-section pydata=pydata.dump tetris
```

On télécharge pyinstxtractor.py :

```
git clone https://github.com/extremecoders-re/pyinstxtractor.git
```

On l'utilise sur le fichier pydata.dump, cela extrait le code source du fichier compilé.

```
python pyinstxtractor.py pydata.dump
```

On obtient le résultat suivant :

```
   ▶   ~/Do/chall_r/tetris/dist ▶ python pyinstxtractor.py pydata.dump     ◀ ✔
[+] Processing pydata.dump
[+] Pyinstaller version: 2.1+
[+] Python version: 3.10
[+] Length of package: 45965488 bytes
[+] Found 165 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: pyi_rth_pkgres.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
[+] Possible entry point: pyi_rth_pkgutil.pyc
[+] Possible entry point: tetris.pyc
[+] Found 418 files in PYZ archive
[+] Successfully extracted pyinstaller archive: pydata.dump

You can now use a python decompiler on the pyc files within the extracted directory
   ▶   ~/Do/chall_r/tetris/dist ▶ ls                                        ◀ ✔
pydata.dump            pyinstxtractor     tetris
pydata.dump_extracted  pyinstxtractor.py  tetris.zip
```

```
    ~/Do/chall_r/tetris/dist ▶ cd pydata.dump_extracted                ◀ ✔
    ~/Do/chall_r/t/d/pydata.dump_extracted ▶ ls                        ◀ ✔
base_library.zip                    libpulsecommon-14-e93e2477.0.so
jaraco                              libpulse-simple-9065ac8d.so.0.1.1
ld-linux-x86-64.so.2                libpython3.10.so.1.0
libattr-4f2a9577.so.1.1.0           libquadmath-96973f99.so.0.0.0
libbz2-a273e504.so.1.0.6            libreadline.so.8
libbz2.so.1.0                       libSDL2-2-302c83d0.0.so.0.16.0
libcap-79733b59.so.2.22             libSDL2_image-2-ca388ea9.0.so.0.2.3
libcrypto.so.3                      libSDL2_mixer-2-d009651f.0.so.0.2.2
libdbus-1-5d678da9.so.3.14.14       libSDL2_ttf-2-cc75fba4.0.so.0.14.1
libdw-0-780627ce.176.so             libselinux-0922c95c.so.1
lib-dynload                         libsndfile-72e2e06b.so.1.0.31
libelf-0-3d9de92a.176.so            libssl.so.3
libexpat.so.1                       libstdc++.so.6
libffi.so.8                         libsystemd-c4cac0a4.so.0.6.0
libFLAC-1889cd2f.so.8.3.0           libtiff-8d84a4e4.so.5.7.0
libfluidsynth-1a88b3a9.so.3.0.3     libvorbis-3a6647c4.so.0.4.9
libfreetype-9936d4a5.so.6.18.0      libvorbisenc-f97181cf.so.2.0.12
libgcc_s.so.1                       libvorbisfile-b464aa2e.so.3.3.8
libgcrypt-18957bae.so.11.8.2        libwebp-c8fa709c.so.7.1.2
libgfortran-040039e1.so.5.0.0       libX11.so.6
libglib-2.0.so.0                    libXau.so.6
libgomp-a34b3233.so.1.0.0           libXdmcp.so.6
libgpg-error-3f09c3c7.so.0.10.0     libz.so.1
libgthread-2.0.so.0                 markupsafe
libjpeg-988b02a7.so.62.3.0          numpy
liblz4-af1653fb.so.1.8.3            pygame
liblzma-07b5b5c8.so.5.2.2           pyiboot01_bootstrap.pyc
liblzma.so.5                        pyimod01_archive.pyc
libmikmod-6af46a87.so.3.0.0         pyimod02_importers.pyc
libmodplug-c335b2c6.so.1.0.0        pyimod03_ctypes.pyc
libmpg123-e4fca779.so.0.46.3        pyi_rth_inspect.pyc
libncursesw.so.6                    pyi_rth_multiprocessing.pyc
libogg-04548ab3.so.0.8.5            pyi_rth_pkgres.pyc
libopenblas64_p-r0-742d56dc.3.20.so pyi_rth_pkgutil.pyc
libopus-06b786df.so.0.8.0           PYZ-00.pyz
libpcre2-8.so.0                     PYZ-00.pyz_extracted
libpcre-9513aab5.so.1.2.0           struct.pyc
libpng16-67f87fff.so.16.37.0        tetris.pyc
libpulse-76af7338.so.0.23.0
```

On voit un fichier tetris.pyc, on essaye de le lire :

```
    ~/Do/chall_r/t/d/pydata.dump_extracted ▶ strings tetris.pyc
```

On obtient le flag :

```
Calibri
Score : z
NHM2I{Sc0r3-m4x-Br4v0!})
z       GAME OVERz
Appuyer sur ESC pour rejouer)
pygamer-
```