**ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**A blue circle with text

AI-generated content may be incorrect.**

**BÁO CÁO ĐỒ ÁN**

**MÔN NGÔN NGỮ LẬP TRÌNH PYTHON**

**Tên đề tài: Xây dựng ứng dụng trò chơi Tiledom**

**Sinh viên thực hiện:**

**Nguyễn Trần Công Danh - 3123410046  
Nguyễn Trần Khả Huy - 3123410125  
Hoàng Việt Bảo Minh - 3123410213  
Mai Tấn Tài - 3123410315**

**Giảng viên: Lê Tấn Long**

**NHẬN XÉT, ĐÁNH GIÁ CỦA GIẢNG VIÊN**

**LỜI CẢM ƠN**

Em xin gửi lời cảm ơn chân thành và sự tri ân sâu sắc đối với các thầy cô của trường Đại Học Sài Gòn, đặc biệt là các thầy cô ở khoa công nghệ thông tin đã tạo điều kiện cho em tiếp cận và tìm hiểu để hoàn thành đồ án môn học lần này. Và em cũng xin chân thành cảm ơn thầy ***Lê Tấn Long***, giáo viên giảng dạy đã nhiệt tình hướng dẫn chúng em hoàn thành đồ án lần này.

Trong quá trình nghiên cứu và làm bài báo cáo đồ án, do kiến thức cũng như kinh nghiệm thực tế còn nhiều hạn chế nên bài báo cáo không thể tránh khỏi những thiếu xót, chúng em rất mong nhận được ý kiến đóng góp từ thầy cô để em học hỏi được nhiều kĩ năng, kinh nghiệm và sẽ hoàn thành tốt hơn trong những bài báo cáo sắp tới.

Em xin chân thành cảm ơn!

**MỤC LỤC**

[PHẦN I. MỞ ĐẦU 6](#_Toc193668137)

[1. Giới thiệu đề tài 6](#_Toc193668138)

[2. Lý do chọn đề tài 6](#_Toc193668139)

[3. Mục đích – mục tiêu của đề tài 6](#_Toc193668140)

[4. Yêu cầu đồ án 6](#_Toc193668141)

[PHẦN II. GIỚI THIỆU 6](#_Toc193668142)

[1. Giới thiệu về ngôn ngữ lập trình Python 6](#_Toc193668143)

[2. Giới thiệu về thư viện Pygame 6](#_Toc193668144)

[3. Một số module thuộc thư viện Pygame 6](#_Toc193668145)

[4. Giới thiệu về trò chơi Tiledom 6](#_Toc193668146)

[PHẦN III. PHÁT TRIỂN ỨNG DỤNG TRÒ CHƠI TILEDOM 6](#_Toc193668147)

[1. Phần tích trò chơi và tìm phương hướng chuyển thể 6](#_Toc193668148)

[2. Thiết kế các màn hình trò chơi 6](#_Toc193668149)

[2.1. Nền trò chơi 6](#_Toc193668150)

[2.2. Màn hình chính 6](#_Toc193668151)

[2.3. Màn hình các màn chơi 6](#_Toc193668152)

[2.4. Màn hình kết quả 6](#_Toc193668153)

[3. Thực hiện viết mã Python cho trò chơi 6](#_Toc193668154)

**PHẦN I. MỞ ĐẦU**

1. ***Giới thiệu đề tài:***

Tên đề tài: Xây dựng ứng dụng trò chơi Tiledom với thư viện Pygame bằng ngôn ngữ lập trình Python.

1. ***Lý do chọn đề tài:***

* Tính sáng tạo và hấp dẫn
* Tiledom là một trò chơi giải đố có lối chơi đơn giản nhưng gây nghiện, phù hợp với nhiều đối tượng người chơi.
* Việc phát triển một game có tính chiến lược và logic giúp nhóm rèn luyện tư duy lập trình.
* Ứng dụng các kiến thức lập trình
* Dự án giúp nhóm áp dụng kiến thức về Python, đặc biệt là lập trình hướng đối tượng (OOP).
* Tích hợp thư viện hỗ trợ đồ họa như Pygame hoặc Tkinter để thiết kế giao diện trực quan.
* Rèn luyện kỹ năng quản lý dữ liệu, thiết kế thuật toán, và xử lý sự kiện trong game.
* Thực tiễn và tiềm năng phát triển
* Game thể loại match-3 như Tiledom có tính phổ biến cao, có thể mở rộng thêm nhiều cấp độ, chế độ chơi mới.
* Nếu hoàn thiện tốt, sản phẩm có thể được phát triển thành một trò chơi thương mại hoặc đăng tải lên các nền tảng chơi game
* Cải thiện kỹ năng làm việc nhóm
* Dự án yêu cầu phối hợp giữa các thành viên từ thiết kế, lập trình, kiểm thử, giúp nhóm nâng cao kỹ năng làm việc nhóm.
* Sử dụng Git/GitHub để quản lý mã nguồn, rèn luyện kỹ năng thực tế trong phát triển phần mềm.

1. ***Mục đích – mục tiêu của đề tài:***

* Mục đích:
* Phát triển một trò chơi Tiledom hoàn chỉnh bằng Python nhằm giúp người chơi giải trí, rèn luyện tư duy logic thông qua các thử thách trong game.
* Ứng dụng các kiến thức lập trình đã học vào thực tiễn, đặc biệt là lập trình game, xử lý đồ họa, thuật toán tìm kiếm và sắp xếp.
* Rèn luyện kỹ năng làm việc nhóm, phân chia công việc hợp lý và sử dụng công cụ hỗ trợ như GitHub, Trello để quản lý dự án.
* Tạo ra một sản phẩm hoàn thiện có thể mở rộng, phát triển thêm nhiều tính năng mới hoặc tối ưu hiệu suất trong tương lai.
* Mục tiêu:
* Xây dựng một trò chơi Tiledom có giao diện đơn giản, trực quan và dễ sử dụng.
* Đảm bảo game chạy mượt mà, có cơ chế chơi hợp lý và không gặp lỗi nghiêm trọng.

1. ***Yêu cầu đồ án:***

Xây dựng 1 ứng dụng giống Tiledom dựa vào kiến thức đã học về ngôn ngữ lập trình Python.

**PHẦN II. GIỚI THIỆU**

1. ***Giới thiệu về ngôn ngữ lập trình Python:***

Python là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình và là ngôn ngữ lập trình dễ học; được dùng rộng rãi trong phát triển trí tuệ nhân tạo. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu. Vào tháng 7 năm 2018, van Rossum đã từ chức lãnh đạo trong cộng đồng ngôn ngữ Python sau 30 năm làm việc.

Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát bộ nhớ tự động; do vậy nó tương tự như Perl, Ruby, Scheme, Smalltalk, và Tcl. Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

Ban đầu, Python được phát triển để chạy trên nền Unix. Nhưng rồi theo thời gian, Python dần mở rộng sang mọi hệ điều hành từ MS-DOS đến Mac OS, OS/2, Windows, Linux và các hệ điều hành khác thuộc họ Unix. Mặc dù sự phát triển của Python có sự đóng góp của rất nhiều cá nhân, nhưng Guido van Rossum hiện nay vẫn là tác giả chủ yếu của Python. Ông giữ vai trò chủ chốt trong việc quyết định hướng phát triển của Python.

Python luôn được xếp hạng vào những ngôn ngữ lập trình phổ biến nhất.

1. ***Giới thiệu về thư viện Python:***

Pygame là một tập hợp các module Python được thiết kế để viết trò chơi điện tử. Pygame bổ sung chức năng lên trên thư viện SDL. Điều này cho phép ta tạo các trò chơi và chương trình đa phương tiện đầy đủ tính năng bằng ngôn ngữ Python.

Pygame có tính di động cao và chạy trên hầu hết mọi nền tảng và hệ điều hành.

Pygame miễn phí. Được phát hành theo giấy phép LGPL, ta có thể tạo mã nguồn mở, phần mềm miễn phí, phần mềm chia sẻ và trò chơi thương mại với nó.

1. ***Một số module thuộc thư viện Pygame:***

|  |  |
| --- | --- |
| Module | Công dụng |
| Pygame.display | Điều khiển cửa sổ hiển thị và màn hình |
| Pygame.draw | Vẽ hình |
| Pygame.event | Tương tác với các sự kiện và các hàng đợi |
| Pygame.font | Tải phông chữ và hiển thị hình ảnh |
| Pygame.image | Tải và hiển thị hình ảnh |
| Pygame.mixer | Tải và phát âm thanh |
| Pygame.time | Theo dõi thời gian (hỗ trợ tạo hiệu ứng trong trò chơi) |

*Một số lớp cơ bản:*

* + pygame.Color : Đại diện cho một màu sắc, gồm 4 thuộc tính thể hiện cho 4 yếu tố màu là đỏ (r), lục (g), lam (b) và độ trong suốt (a). Các đối tượng của lớp này được dùng để truyền và các hàm có hỗ trợ trong các module của thư viện Pygame đề xác định màu cần vẽ.
  + pygame.Surface : Đại diện cho một bề mặt hay khung vẽ cho phép ta vẽ hình lên nó. Đối tượng của lớp này có thể là khung màn hình (là giá trị trả về của hàm pygame.display.set\_mode()), hay khung được người lập trình định nghĩa để chứa một nhóm hình được vẽ trước mà về sau có thể vẽ lên khung màn hình. Lớp cung cấp hàm blit() cho phép vẽ khung này lên khung khác hay lên màn hình.
  + pygame.Rect : Là lớp cho phép tạo các đối tượng lưu trữ tọa độ của các vật thể trên màn hình. Gồm một số thuộc tính có thể sử dụng như: x, y, w, h, top, left, bottom, right, size, width, height.
  + pygame.event.Event : Là lớp đại diện cho các sự kiện xảy ra trong chương trình, chẵng hạn như thoát chương trình, người dùng nhập bàn phím, chuột di chuyển hay cả việc kích thước màn hình thay đổi.
  + pygame.font.Font : Là lớp hỗ trợ việc tải phông chữ và có hàm cho phép tạo ra pygame.Surface mới và vẽ chữ trên nó cho phông chữ hiện tại.

*Một số hàm cơ bản:*

* + pygame.init() : là hàm giúp khởi tạo, tải các thành phần của thư viện Pygame vào ứng dụng trò chơi của ta.
  + pygame.quit() : là hàm giúp ta hủy khởi tạo các thành phần của thư viện của Pygame, giúp ta thoát ứng dụng một cách an toàn.
  + pygame.display.set\_mode() : là hàm giúp khởi tạo cửa sổ hoặc màn hình để hiển thị.
  + pygame.display.flip() : là hàm giúp cập nhật lại toàn bộ màn hình.
  + pygame.display.update() : là hàm giúp cập nhật một phần màn hình nếu ta cung cấp các đối tượng thuộc lớp pygame.Rect cần cập nhật, ngược lại nó sẽ cập nhật toàn bộ màn hình.
  + pygame.display.Info() : là hàm giúp thu thập các thông số của màn hình hiển thị cũng như cửa sổ ứng dụng hiện tại.
  + pygame.display.set\_caption() : Là hàm giúp thay đổi tiêu đề của cửa sổ ứng dụng.
  + pygame.draw.rect() : Là hàm dùng để vẽ một hình chữ nhật lên các đối tượng thuộc lớp pygame.Surface.
  + pygame.draw.circle() : Là hàm dùng để vẽ một hình chữ nhật lên các đối tượng thuộc lớp pygame.Surface.
  + pygame.event.get() : Là hàm giúp lấy ra toàn bộ các sự kiện trong hàng đợi sự kiện của ứng dụng.
  + pygame.font.SysFont() : là hàm giúp tải phông chữ hệ thống từ tên phông chữ hệ thống.
  + pygame.font.Font.render() : là hàm giúp tạo ra pygame.Surface và vẽ chữ trên nó cho phông chữ hiện tại.
  + pygame.image.load() : là hàm giúp tạo ra pygame.Surface và vẽ ảnh đó lên.
  + pygame.time.Clock.tick() : là hàm giúp tính toán và trả về thời gian giữa hai lần vẽ lên màn hình.

1. ***Giới thiệu về trò chơi Tiledom:***

Tiledom là một trò chơi giải đố (puzzle) thú vị, thuộc thể loại match-3, nơi người chơi phải tìm và ghép ba ô giống nhau để loại bỏ chúng khỏi bảng chơi. Trò chơi này có cơ chế đơn giản nhưng đầy thử thách, đòi hỏi sự tập trung và chiến lược để giành chiến thắng.

* + Cách chơi:
    - Người chơi sẽ được cung cấp một bảng chứa nhiều ô có hình dạng hoặc biểu tượng khác nhau.
    - Nhiệm vụ là chọn tối đa **ba ô giống nhau** để đưa vào khay chứa.
    - Khi ba ô giống nhau nằm trong khay, chúng sẽ tự động biến mất.
    - Nếu khay chứa bị đầy mà chưa tìm được đủ bộ ba, người chơi sẽ thua.
    - Trò chơi tiếp tục cho đến khi người chơi xóa hết các ô trên bảng.

A screenshot of a game

AI-generated content may be incorrect.

Hình 2.1: Hình ảnh minh họa một màn chơi trong Tiledom

**PHẦN III. Phát triển ứng dụng trò chơi Tiledom**

1. ***Phân tích trò chơi và tìm phương hướng chuyển thể:***
2. Phân tích trò chơi Tiledom:

Gameplay cơ bản:

* + Màn chơi sẽ bao gồm một lưới các ô vuông (grid).
  + Các ô vuông có thể chứa các khối hình khác nhau (như các hình vuông, chữ nhật hoặc các khối ghép).
  + Người chơi sẽ di chuyển các khối hoặc đổi chỗ các khối để tạo ra một tổ hợp hợp lệ hoặc đạt được mục tiêu đề ra.
  + Có thể có các tính năng bổ sung như di chuyển hạn chế, điểm số, mức độ khó tăng dần, hoặc các vật phẩm hỗ trợ.

Đặc điểm của trò chơi:

* + Khối và lưới: Một lưới hình vuông hoặc chữ nhật là nơi người chơi sẽ thực hiện các hành động. Các khối có thể được biểu thị bằng các hình ảnh hoặc màu sắc khác nhau.
  + Điều khiển: Thông qua các sự kiện chuột hoặc bàn phím, người chơi có thể di chuyển các khối trong lưới.
  + Mục tiêu: Tùy vào mỗi cấp độ, mục tiêu có thể là ghép một số khối hình cụ thể hoặc đạt điểm số tối đa.
  + Tính năng bổ sung: Có thể có các tính năng như "undo", "reset", hoặc tăng dần độ khó với các loại khối phức tạp hơn.

1. Phương hướng chuyển thể game Tiledom trong Python với Pygame:

Tạo lưới trò chơi:

* + Sử dụng Pygame để tạo lưới, mỗi ô trong lưới có thể là một đối tượng hoặc một vùng có thể chứa các khối.
  + Lưới có thể là một mảng 2D, với mỗi phần tử là một ô vuông chứa một khối hoặc rỗng.

Các đối tượng trong trò chơi:

* + Khối: Mỗi khối sẽ có các thuộc tính như màu sắc, hình dạng, hoặc giá trị điểm. Bạn có thể tạo ra lớp (class) cho các khối này trong Python.
  + Lưới: Lưới sẽ chứa một số ô (cells) mà mỗi ô có thể chứa một khối. Bạn sẽ cần lập trình để khối có thể di chuyển trong lưới và kiểm tra điều kiện trùng khớp.

Quy trình game:

* + Vẽ lưới và các khối: Mỗi khối cần được vẽ trong mỗi ô của lưới, bạn có thể sử dụng các hàm của Pygame để vẽ hình vuông hoặc các hình khác cho mỗi khối.
  + Xử lý sự kiện: Bạn cần viết code để xử lý các sự kiện như di chuyển chuột, nhấn phím để thay đổi vị trí các khối.
  + Logic trận đấu: Sau khi người chơi thực hiện các bước di chuyển, bạn cần kiểm tra điều kiện thắng thua hoặc tiếp tục trò chơi.

Các tính năng bổ sung:

* + Điểm số: Bạn có thể thêm hệ thống điểm số để người chơi có thể theo dõi tiến độ của mình.
  + Cấp độ: Tạo các cấp độ với độ khó tăng dần, có thể bằng cách thêm các khối phức tạp hoặc yêu cầu di chuyển nhiều hơn.
  + Hiệu ứng âm thanh: Thêm âm thanh khi người chơi hoàn thành một lượt hoặc đạt được mục tiêu.

Chuyển thể sang môi trường khác:

* + Web: Nếu muốn, bạn có thể chuyển thể game từ Python sang môi trường web sử dụng JavaScript và HTML5 canvas.
  + Ứng dụng di động: Sử dụng công cụ như Kivy (một framework Python) để chuyển game thành ứng dụng di động.

1. ***Thiết kế các màn hình trò chơi:***

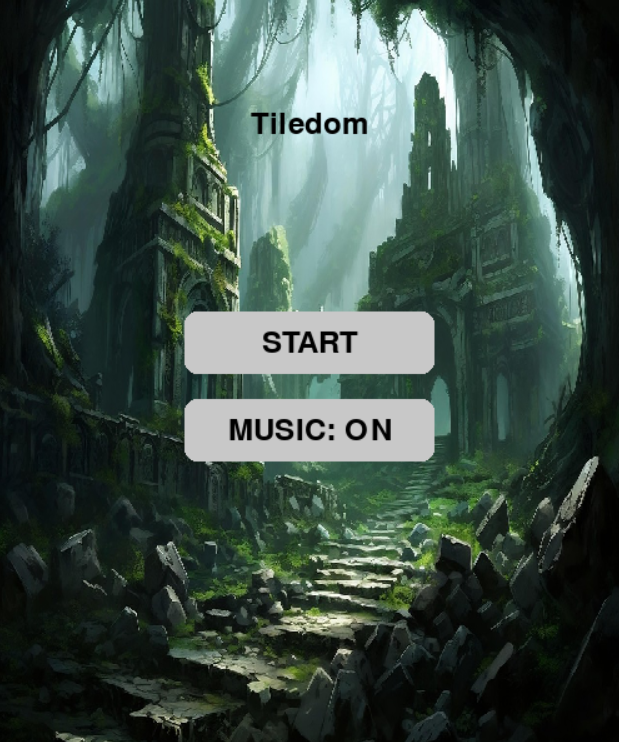
***2.1. Nền trò chơi:***

-Nền cơ sở (base background) là một hình họa tiết bất kỳ.



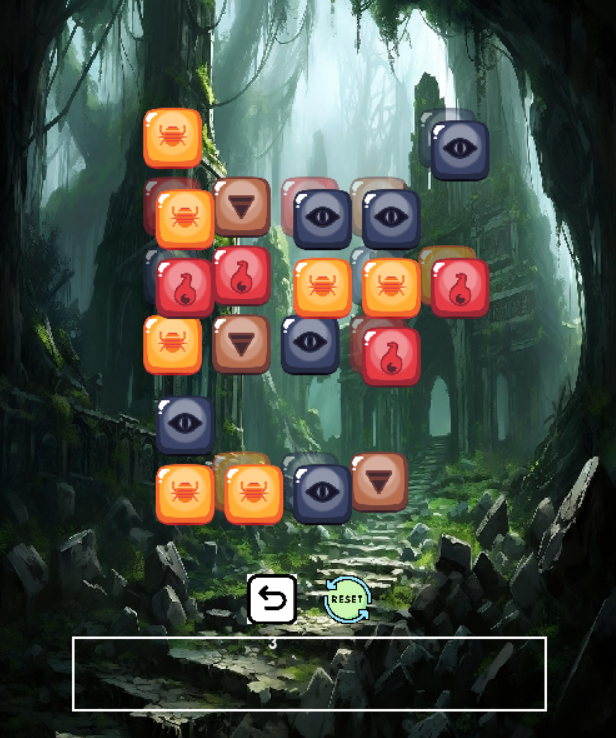
Hình 2.1: Hình nền của trò chơi

***2.2. Màn hình chính:***



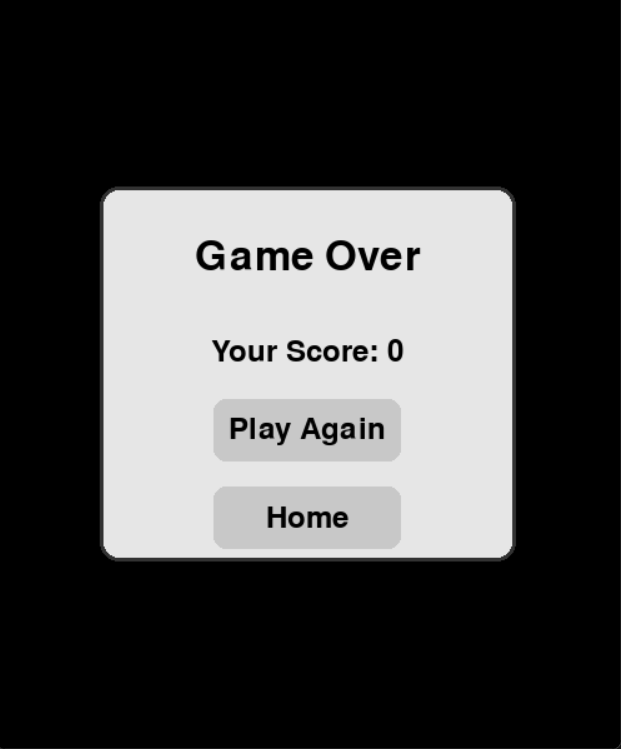
Hình 2.2: Màn hình chính của trò chơi

***2.3. Màn hình các màn chơi:***

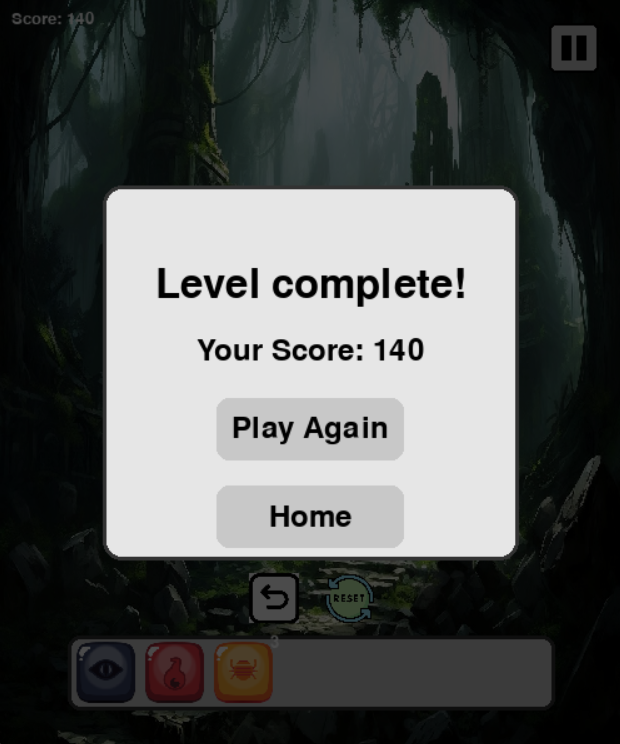


Hình 2.3: Màn hình màn chơi

***2.4. Màn hình kết quả:***



Hình 2.4.1: Màn hình khi thua



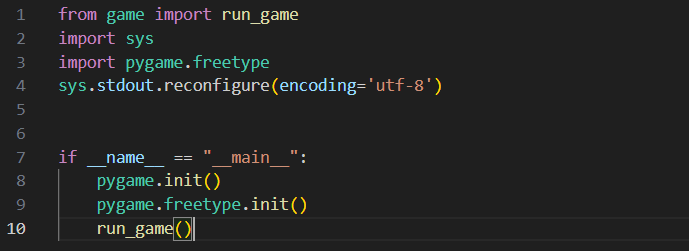
Hình 2.4.2: Màn hình khi hoàn thành trò chơi

1. ***Thực hiện viết mã Python cho trò chơi:***

* Chuẩn bị:
  + Môi trường viết mã: Visual Studio Code
  + Môi trường chạy mã: Terminal, Python3
  + Thư viện cần cài đặt bổ sung: Pygame
  + Tạo cây thư mục và một số file cần thiết:
    - ./assets, ./assets/images, ./assets/sounds, ./src, ./src/main.py, ./src/game.py, ./src/player.py, ./src/settings.py
* Các hình ảnh của game:



**Bắt đầu với ./main.py:**



**Tiếp theo là ./game.py:**

import pygame

import random

import os

import math

from settings import draw\_button, screen, resized\_background, SCREEN\_WIDTH, SCREEN\_HEIGHT, home\_page, reset\_game, font, image\_list, play\_click\_sound, play\_match\_sound, play\_complete\_sound, play\_gameover\_sound, base\_dir, square\_size, spacing

from player import squares, update\_grid\_position

# Khởi tạo Pygame

pygame.init()

# Tải hình ảnh

UNDO\_IMAGE\_PATH = os.path.join(base\_dir, "images", "undo.jpg")

RESET\_IMAGE\_PATH = os.path.join(base\_dir, "images", "Reset.jpg")

try:

    undo\_image = pygame.image.load(UNDO\_IMAGE\_PATH).convert\_alpha()

    reset\_image = pygame.image.load(RESET\_IMAGE\_PATH).convert\_alpha()

except pygame.error as e:

    print(f"Không thể tải hình ảnh: {e}")

    undo\_image = None

    reset\_image = None

# Cấu hình nút tròn

BUTTON\_RADIUS = 20

BUTTON\_SPACING = 20

CENTER\_X = SCREEN\_WIDTH // 2

BUTTON\_Y = SCREEN\_HEIGHT - 120

# Cấu hình chữ số lượt và điểm

COUNTER\_TEXT\_COLOR = (255, 255, 255)

COUNTER\_FONT\_SIZE = 20

counter\_font = pygame.font.Font(None, COUNTER\_FONT\_SIZE)

class Square:

    def \_\_init\_\_(self, image, rect, layer=1, col=0, row=0):

        self.image = image

        self.rect = pygame.Rect(rect)

        self.layer = layer

        self.col = col

        self.row = row

        self.original\_rect = rect.copy()  # Lưu tọa độ gốc

    def draw(self, is\_top\_layer=True):

        scaled\_image = pygame.transform.scale(self.image, (square\_size, square\_size))

        if not is\_top\_layer:

            scaled\_image.set\_alpha(128)

        screen.blit(scaled\_image, self.rect.topleft)

class SquareManager:

    def \_\_init\_\_(self):

        self.selected\_squares = []

        self.max\_selected = 7

        self.history = []

        self.undo\_limit = 3

        self.undo\_count = 0

        self.score = 0

    def add\_square(self, square):

        if square not in self.selected\_squares and len(self.selected\_squares) < self.max\_selected:

            square\_data = Square(square.image, square.original\_rect.copy() if hasattr(square, 'original\_rect') else square.rect.copy(),

                                layer=square.layer if hasattr(square, 'layer') else 1,

                                col=square.col if hasattr(square, 'col') else 0,

                                row=square.row if hasattr(square, 'row') else 0)

            self.history.append(("add", square\_data))

            self.selected\_squares.append(square)

            self.update\_selected\_position()

            print(f"Added square: original\_pos=({square\_data.rect.x},{square\_data.rect.y}), history\_length={len(self.history)}")

            if len(self.selected\_squares) == self.max\_selected and not self.check\_match():

                print("Khung đầy và không có match, thua game!")

                return False

            self.check\_match()

            return True

        print(f"Cannot add square: already\_in\_list={square in self.selected\_squares}, frame\_full={len(self.selected\_squares) >= self.max\_selected}")

        return True

    def update\_selected\_position(self):

        frame\_width = 7 \* square\_size + 6 \* 5 + 10

        frame\_x = (SCREEN\_WIDTH - frame\_width) // 2

        frame\_y = SCREEN\_HEIGHT - (square\_size + 40)

        start\_x = frame\_x + 5

        start\_y = frame\_y + 5

        for i, square in enumerate(self.selected\_squares):

            square.rect.x = start\_x + i \* (square\_size + 5)

            square.rect.y = start\_y

            square.rect.width = square\_size

            square.rect.height = square\_size

        print(f"Updated selected\_squares positions: {len(self.selected\_squares)} squares")

    def check\_match(self):

        img\_count = {}

        for square in self.selected\_squares:

            img\_count[square.image] = img\_count.get(square.image, 0) + 1

        matched\_images = {img for img, count in img\_count.items() if count >= 3}

        if matched\_images:

            play\_match\_sound()

            removed\_squares = [Square(sq.image, sq.rect.copy(), sq.layer, sq.col, sq.row) for sq in self.selected\_squares if sq.image in matched\_images]

            self.history.append(("remove", removed\_squares))

            self.selected\_squares = [sq for sq in self.selected\_squares if sq.image not in matched\_images]

            self.update\_selected\_position()

            self.score += 10

            self.undo\_count = max(0, self.undo\_count - len(removed\_squares))

            print(f"Match found: Removed {len(removed\_squares)} squares, score={self.score}, history\_length={len(self.history)}")

            return True

        print(f"No match found: selected\_squares={len(self.selected\_squares)}")

        return False

    def draw\_selected\_squares(self):

        for square in self.selected\_squares:

            square.draw(is\_top\_layer=True)

    def undo(self):

        if self.history and self.undo\_count < self.undo\_limit:

            last\_action, data = self.history.pop()

            print(f"Undoing action: {last\_action}, undo\_count={self.undo\_count}, history\_length={len(self.history)}")

            if last\_action == "add":

                if self.selected\_squares:

                    last\_square = self.selected\_squares[-1]

                    if last\_square.image == data.image:

                        self.selected\_squares.pop()

                        self.update\_selected\_position()

                        pos = (data.col, data.row, data.layer)

                        if not any(sq["col"] == data.col and sq["row"] == data.row and sq["layer"] == data.layer for sq in squares):

                            restored\_rect = data.rect.copy()

                            squares.append({

                                "rect": restored\_rect,

                                "image": data.image,

                                "layer": data.layer,

                                "col": data.col,

                                "row": data.row

                            })

                            print(f"Undo add: Restored square, layer={data.layer}, col={data.col}, row={data.row}, rect={restored\_rect.topleft}")

                        else:

                            print(f"Undo add: Position (col={data.col}, row={data.row}, layer={data.layer}) already occupied")

                        self.undo\_count += 1

                    else:

                        print("Undo add: Last square does not match data")

                else:

                    print("Undo add: No squares in selected\_squares")

            elif last\_action == "remove":

                restored\_squares = [Square(sq.image, sq.rect.copy(), sq.layer, sq.col, sq.row) for sq in data]

                self.selected\_squares.extend(restored\_squares)

                self.update\_selected\_position()

                self.undo\_count += len(restored\_squares)

                print(f"Undo remove: Restored {len(restored\_squares)} squares")

        else:

            print(f"Cannot undo: history\_empty={not self.history}, undo\_limit\_reached={self.undo\_count >= self.undo\_limit}")

def draw\_circular\_button(screen, image, center\_x, center\_y, radius, active=True, counter\_text=None):

    if image:

        scaled\_image = pygame.transform.scale(image, (radius \* 2, radius \* 2))

        rect = scaled\_image.get\_rect(center=(center\_x, center\_y))

        mask = pygame.mask.from\_surface(scaled\_image)

        surface\_to\_blit = pygame.Surface(rect.size, pygame.SRCALPHA)

        for x in range(rect.width):

            for y in range(rect.height):

                if mask.get\_at((x, y)):

                    surface\_to\_blit.set\_at((x, y), scaled\_image.get\_at((x, y)))

        screen.blit(surface\_to\_blit, rect)

    if active and counter\_text is not None:

        text\_surface = counter\_font.render(counter\_text, True, COUNTER\_TEXT\_COLOR)

        text\_rect = text\_surface.get\_rect(center=(center\_x, center\_y + radius + 15))

        screen.blit(text\_surface, text\_rect)

    elif not active and counter\_text is not None:

        text\_surface = counter\_font.render(counter\_text, True, (128, 128, 128))

        text\_rect = text\_surface.get\_rect(center=(center\_x, center\_y + radius + 15))

        screen.blit(text\_surface, text\_rect)

def draw\_ui():

    frame\_width = 7 \* square\_size + 6 \* 5 + 10

    frame\_height = square\_size + 10

    frame\_x = (SCREEN\_WIDTH - frame\_width) // 2

    frame\_y = SCREEN\_HEIGHT - frame\_height - 30

    shadow\_surface = pygame.Surface((frame\_width + 10, frame\_height + 10), pygame.SRCALPHA)

    pygame.draw.rect(shadow\_surface, (0, 0, 0, 100), (5, 5, frame\_width, frame\_height), border\_radius=10)

    screen.blit(shadow\_surface, (frame\_x - 5, frame\_y - 5))

    pygame.draw.rect(screen, (200, 200, 200), (frame\_x, frame\_y, frame\_width, frame\_height), border\_radius=10)

    pygame.draw.rect(screen, (50, 50, 50), (frame\_x, frame\_y, frame\_width, frame\_height), 3, border\_radius=10)

def lose():

    global game\_state

    game\_state = "lose"

    play\_gameover\_sound()

square\_manager = SquareManager()

reset\_available = True

def generate\_random\_squares(num\_squares=45):

    global squares

    squares = []

    available\_positions = set()

    grid\_width = 5

    grid\_height = 6

    max\_layers = 2

    for r in range(grid\_height):

        for c in range(grid\_width):

            for layer in range(max\_layers):

                available\_positions.add((c, r, layer))

    image\_choices = random.choices(image\_list, k=num\_squares)

    grid\_pixel\_width = grid\_width \* square\_size + (grid\_width - 1) \* spacing

    grid\_pixel\_height = grid\_height \* square\_size + (grid\_height - 1) \* spacing

    offset\_x = (SCREEN\_WIDTH - grid\_pixel\_width) // 2

    offset\_y = (SCREEN\_HEIGHT - grid\_pixel\_height) // 2 - 50

    layer\_offset = 10

    for img in image\_choices:

        if not available\_positions:

            break

        position = random.choice(list(available\_positions))

        col, row, layer = position

        x = offset\_x + col \* (square\_size + spacing)

        y = offset\_y + row \* (square\_size + spacing)

        if layer == 1:

            x += layer\_offset

            y += layer\_offset

        squares.append({

            "rect": pygame.Rect(x, y, square\_size, square\_size),

            "image": img,

            "layer": layer,

            "col": col,

            "row": row

        })

        available\_positions.discard(position)

    print(f"Generated {len(squares)} squares")

def move\_square\_to\_selected(square\_data, target\_rect, squares\_state, animation\_duration=500):

    global is\_animating

    is\_animating = True

    start\_rect = pygame.Rect(square\_data["rect"])

    end\_rect = pygame.Rect(target\_rect)

    start\_time = pygame.time.get\_ticks()

    moving\_square = Square(square\_data["image"], start\_rect)

    control\_point\_x = (start\_rect.x + end\_rect.x) / 2

    control\_point\_y = min(start\_rect.y, end\_rect.y) - 100

    max\_layer\_at\_position = {}

    for sq in squares\_state:

        pos = (sq["col"], sq["row"])

        current\_layer = sq["layer"]

        if pos not in max\_layer\_at\_position or current\_layer > max\_layer\_at\_position[pos]:

            max\_layer\_at\_position[pos] = current\_layer

    running = True

    while running:

        current\_time = pygame.time.get\_ticks()

        time\_elapsed = current\_time - start\_time

        if time\_elapsed >= animation\_duration:

            moving\_square.rect = end\_rect

            running = False

            for i in range(3):

                screen.blit(resized\_background, (0, 0))

                for sq in sorted(squares\_state, key=lambda x: x["layer"]):

                    if sq != square\_data:

                        scaled\_image = pygame.transform.scale(sq["image"], (square\_size, square\_size))

                        pos = (sq["col"], sq["row"])

                        max\_layer = max\_layer\_at\_position.get(pos, 0)

                        if sq["layer"] == 0 and max\_layer == 1:

                            scaled\_image.set\_alpha(128)

                        else:

                            scaled\_image.set\_alpha(255)

                        screen.blit(scaled\_image, sq["rect"].topleft)

                draw\_ui()

                square\_manager.draw\_selected\_squares()

                moving\_square.rect.x += (1 if i % 2 == 0 else -1) \* 5

                scaled\_image = pygame.transform.scale(moving\_square.image, (square\_size, square\_size))

                screen.blit(scaled\_image, moving\_square.rect.topleft)

                pygame.display.flip()

                pygame.time.wait(50)

            moving\_square.rect.x = end\_rect.x

        else:

            progress = time\_elapsed / animation\_duration

            t = progress

            x = (1 - t) \*\* 2 \* start\_rect.x + 2 \* (1 - t) \* t \* control\_point\_x + t \*\* 2 \* end\_rect.x

            y = (1 - t) \*\* 2 \* start\_rect.y + 2 \* (1 - t) \* t \* control\_point\_y + t \*\* 2 \* end\_rect.y

            moving\_square.rect.x = int(x)

            moving\_square.rect.y = int(y)

            scale = 1.0 + 0.2 \* math.sin(math.pi \* progress)

            scaled\_size = int(square\_size \* scale)

            scaled\_image = pygame.transform.scale(moving\_square.image, (scaled\_size, scaled\_size))

            rotation = 360 \* progress

            rotated\_image = pygame.transform.rotate(scaled\_image, rotation)

            rotated\_rect = rotated\_image.get\_rect(center=(moving\_square.rect.centerx, moving\_square.rect.centery))

            alpha = 255 \* (1 - 0.3 \* math.sin(math.pi \* progress))

            rotated\_image.set\_alpha(int(alpha))

            screen.blit(resized\_background, (0, 0))

            for sq in sorted(squares\_state, key=lambda x: x["layer"]):

                if sq != square\_data:

                    scaled\_image = pygame.transform.scale(sq["image"], (square\_size, square\_size))

                    pos = (sq["col"], sq["row"])

                    max\_layer = max\_layer\_at\_position.get(pos, 0)

                    if sq["layer"] == 0 and max\_layer == 1:

                        scaled\_image.set\_alpha(128)

                    else:

                        scaled\_image.set\_alpha(255)

                    screen.blit(scaled\_image, sq["rect"].topleft)

            draw\_ui()

            square\_manager.draw\_selected\_squares()

            screen.blit(rotated\_image, rotated\_rect.topleft)

            pygame.display.flip()

            pygame.time.Clock().tick(120)

    is\_animating = False

def reset\_board():

    global squares, reset\_available

    if reset\_available:

        play\_click\_sound()

        print("Resetting board")

        squares = []

        num\_squares = max(45 - len(square\_manager.selected\_squares), 20)

        generate\_random\_squares(num\_squares)

        reset\_available = False

        print(f"Reset complete: Generated {len(squares)} new squares, kept {len(square\_manager.selected\_squares)} selected squares")

def run\_game():

    global game\_state, squares, reset\_available, is\_animating

    running = True

    dragging = None

    offset\_x, offset\_y = 0, 0

    mouse\_moved = False

    game\_state = "home"

    selected\_a\_square = None

    dragging\_original\_index = None

    is\_animating = False

    mouse\_start\_pos = None

    mouse\_move\_threshold = 5

    undo\_button\_center\_x = CENTER\_X - BUTTON\_RADIUS - BUTTON\_SPACING // 2

    reset\_button\_center\_x = CENTER\_X + BUTTON\_RADIUS + BUTTON\_SPACING // 2

    undo\_button\_center\_y = BUTTON\_Y

    reset\_button\_center\_y = BUTTON\_Y

    undo\_button\_rect = pygame.Rect(undo\_button\_center\_x - BUTTON\_RADIUS, undo\_button\_center\_y - BUTTON\_RADIUS, BUTTON\_RADIUS \* 2, BUTTON\_RADIUS \* 2)

    reset\_button\_rect = pygame.Rect(reset\_button\_center\_x - BUTTON\_RADIUS, reset\_button\_center\_y - BUTTON\_RADIUS, BUTTON\_RADIUS \* 2, BUTTON\_RADIUS \* 2)

    while running:

        screen.blit(resized\_background, (0, 0))

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                running = False

            if game\_state == "home":

                reset\_game()

                home\_page()

                generate\_random\_squares(45)

                square\_manager.undo\_count = 0

                square\_manager.score = 0

                reset\_available = True

                game\_state = "play"

            elif game\_state == "play":

                if event.type == pygame.MOUSEBUTTONDOWN and dragging is None and not is\_animating:

                    mouse\_pos = event.pos

                    mouse\_moved = False

                    mouse\_start\_pos = mouse\_pos

                    if reset\_image and reset\_button\_rect.collidepoint(mouse\_pos) and reset\_available:

                        print("Calling reset\_board from MOUSEBUTTONDOWN")

                        reset\_board()

                        continue

                    if undo\_image and undo\_button\_rect.collidepoint(mouse\_pos) and square\_manager.undo\_count < square\_manager.undo\_limit:

                        print("Calling undo from MOUSEBUTTONDOWN")

                        play\_click\_sound()

                        square\_manager.undo()

                        screen.blit(resized\_background, (0, 0))

                        max\_layer\_at\_position = {}

                        for square in squares:

                            pos = (square["col"], square["row"])

                            current\_layer = square["layer"]

                            if pos not in max\_layer\_at\_position or current\_layer > max\_layer\_at\_position[pos]:

                                max\_layer\_at\_position[pos] = current\_layer

                        for square in sorted(squares, key=lambda x: x["layer"]):

                            pos = (square["col"], square["row"])

                            max\_layer = max\_layer\_at\_position.get(pos, 0)

                            scaled\_image = pygame.transform.scale(square["image"], (square\_size, square\_size))

                            if square["layer"] == 0 and max\_layer == 1:

                                scaled\_image.set\_alpha(128)

                            else:

                                scaled\_image.set\_alpha(255)

                            screen.blit(scaled\_image, square["rect"].topleft)

                            print(f"Drawing square: rect={square['rect'].topleft}, layer={square['layer']}, col={square['col']}, row={square['row']}")

                        draw\_ui()

                        square\_manager.draw\_selected\_squares()

                        if undo\_image:

                            draw\_circular\_button(screen, undo\_image, undo\_button\_center\_x, undo\_button\_center\_y, BUTTON\_RADIUS, square\_manager.undo\_count < square\_manager.undo\_limit, str(square\_manager.undo\_limit - square\_manager.undo\_count))

                        if reset\_image:

                            draw\_circular\_button(screen, reset\_image, reset\_button\_center\_x, reset\_button\_center\_y, BUTTON\_RADIUS, reset\_available)

                        score\_text = counter\_font.render(f"Score: {square\_manager.score}", True, COUNTER\_TEXT\_COLOR)

                        screen.blit(score\_text, (10, 10))

                        pygame.display.flip()

                        continue

                    max\_layer\_at\_position = {}

                    for square in squares:

                        pos = (square["col"], square["row"])

                        current\_layer = square["layer"]

                        if pos not in max\_layer\_at\_position or current\_layer > max\_layer\_at\_position[pos]:

                            max\_layer\_at\_position[pos] = current\_layer

                    colliding\_squares = [

                        (i, square) for i, square in enumerate(squares)

                        if square["rect"].collidepoint(mouse\_pos)

                    ]

                    if colliding\_squares:

                        valid\_squares = [

                            (i, square) for i, square in colliding\_squares

                            if square["layer"] == max\_layer\_at\_position.get((square["col"], square["row"]), -1)

                        ]

                        if valid\_squares:

                            i, square = max(valid\_squares, key=lambda x: x[1]["layer"])

                            dragging = Square(square["image"], square["rect"].copy(),

                                            layer=square["layer"], col=square["col"], row=square["row"])

                            dragging\_original\_index = i

                            offset\_x = mouse\_pos[0] - square["rect"].x

                            offset\_y = mouse\_pos[1] - square["rect"].y

                            selected\_a\_square = {

                                "rect": square["rect"].copy(),

                                "image": square["image"],

                                "layer": square["layer"],

                                "col": square["col"],

                                "row": square["row"]

                            }

                if event.type == pygame.MOUSEMOTION and dragging:

                    mouse\_pos = event.pos

                    distance = ((mouse\_pos[0] - mouse\_start\_pos[0]) \*\* 2 + (mouse\_pos[1] - mouse\_start\_pos[1]) \*\* 2) \*\* 0.5

                    if distance > mouse\_move\_threshold:

                        mouse\_moved = True

                    dragging.rect.x = mouse\_pos[0] - offset\_x

                    dragging.rect.y = mouse\_pos[1] - offset\_y

                if event.type == pygame.MOUSEBUTTONUP and dragging:

                    if not mouse\_moved:

                        if len(square\_manager.selected\_squares) < square\_manager.max\_selected and selected\_a\_square is not None and dragging\_original\_index is not None and 0 <= dragging\_original\_index < len(squares):

                            play\_click\_sound()

                            frame\_width = 7 \* square\_size + 6 \* 5 + 10

                            frame\_x = (SCREEN\_WIDTH - frame\_width) // 2

                            target\_x = frame\_x + 5 + len(square\_manager.selected\_squares) \* (square\_size + 5)

                            target\_y = SCREEN\_HEIGHT - (square\_size + 35)

                            target\_rect = pygame.Rect(target\_x, target\_y, square\_size, square\_size)

                            move\_square\_to\_selected(selected\_a\_square, target\_rect, squares.copy())

                            del squares[dragging\_original\_index]

                            new\_square = Square(selected\_a\_square["image"], target\_rect,

                                               layer=selected\_a\_square["layer"],

                                               col=selected\_a\_square["col"],

                                               row=selected\_a\_square["row"])

                            new\_square.original\_rect = selected\_a\_square["rect"].copy()

                            if not square\_manager.add\_square(new\_square):

                                lose()

                    dragging = None

                    selected\_a\_square = None

                    dragging\_original\_index = None

                    mouse\_start\_pos = None

                if not squares and event.type == pygame.MOUSEBUTTONDOWN:

                    if box\_rect.collidepoint(mouse\_pos):

                        game\_state = "home"

            elif game\_state == "lose":

                overlay = pygame.Surface((SCREEN\_WIDTH, SCREEN\_HEIGHT), pygame.SRCALPHA)

                overlay.fill((0, 0, 0, 180))

                screen.blit(overlay, (0, 0))

                box\_rect = pygame.Rect(SCREEN\_WIDTH // 6, SCREEN\_HEIGHT // 4, SCREEN\_WIDTH \* 2 // 3, SCREEN\_HEIGHT \* 1 // 2)

                # Thêm bóng cho khung

                shadow\_surface = pygame.Surface((box\_rect.width + 10, box\_rect.height + 10), pygame.SRCALPHA)

                pygame.draw.rect(shadow\_surface, (0, 0, 0, 100), (5, 5, box\_rect.width, box\_rect.height), border\_radius=15)

                screen.blit(shadow\_surface, (box\_rect.x - 5, box\_rect.y - 5))

                # Vẽ khung

                pygame.draw.rect(screen, (230, 230, 230), box\_rect, border\_radius=15)

                pygame.draw.rect(screen, (50, 50, 50), box\_rect, 3, border\_radius=15)

                # Văn bản "Game Over"

                game\_over\_font = pygame.font.Font(None, 48)

                text = game\_over\_font.render("Game Over", True, (0, 0, 0))

                text\_rect = text.get\_rect(center=(box\_rect.centerx, box\_rect.top + 80))

                screen.blit(text, text\_rect)

                # Nút

                button\_width = 150

                button\_height = 50

                button\_spacing = 10

                play\_again\_button = pygame.Rect(

                    box\_rect.centerx - button\_width // 2,

                    box\_rect.centery - button\_height - button\_spacing // 2 + 30,

                    button\_width,

                    button\_height

                )

                home\_button = pygame.Rect(

                    box\_rect.centerx - button\_width // 2,

                    box\_rect.centery + button\_spacing // 2 + 30,

                    button\_width,

                    button\_height

                )

                draw\_button(play\_again\_button, "Play Again")

                draw\_button(home\_button, "Home")

                if event.type == pygame.MOUSEBUTTONDOWN:

                    if play\_again\_button.collidepoint(event.pos):

                        play\_click\_sound()

                        reset\_game()

                        square\_manager.selected\_squares = []

                        square\_manager.history = []

                        square\_manager.undo\_count = 0

                        square\_manager.score = 0

                        reset\_available = True

                        generate\_random\_squares(45)

                        game\_state = "play"

                    elif home\_button.collidepoint(event.pos):

                        play\_click\_sound()

                        game\_state = "home"

        if game\_state == "play":

            squares\_copy = squares.copy()

            max\_layer\_at\_position = {}

            for square in squares\_copy:

                pos = (square["col"], square["row"])

                current\_layer = square["layer"]

                if pos not in max\_layer\_at\_position or current\_layer > max\_layer\_at\_position[pos]:

                    max\_layer\_at\_position[pos] = current\_layer

            for square in sorted(squares\_copy, key=lambda x: x["layer"]):

                pos = (square["col"], square["row"])

                max\_layer = max\_layer\_at\_position.get(pos, 0)

                scaled\_image = pygame.transform.scale(square["image"], (square\_size, square\_size))

                if square["layer"] == 0 and max\_layer == 1:

                    scaled\_image.set\_alpha(128)

                else:

                    scaled\_image.set\_alpha(255)

                screen.blit(scaled\_image, square["rect"].topleft)

            draw\_ui()

            square\_manager.draw\_selected\_squares()

            if undo\_image:

                draw\_circular\_button(screen, undo\_image, undo\_button\_center\_x, undo\_button\_center\_y, BUTTON\_RADIUS, square\_manager.undo\_count < square\_manager.undo\_limit, str(square\_manager.undo\_limit - square\_manager.undo\_count))

            if reset\_image:

                draw\_circular\_button(screen, reset\_image, reset\_button\_center\_x, reset\_button\_center\_y, BUTTON\_RADIUS, reset\_available)

            score\_text = counter\_font.render(f"Score: {square\_manager.score}", True, COUNTER\_TEXT\_COLOR)

            screen.blit(score\_text, (10, 10))

            if not squares:

                overlay = pygame.Surface((SCREEN\_WIDTH, SCREEN\_HEIGHT), pygame.SRCALPHA)

                overlay.fill((0, 0, 0, 180))

                screen.blit(overlay, (0, 0))

                box\_rect = pygame.Rect(SCREEN\_WIDTH // 6, SCREEN\_HEIGHT // 4, SCREEN\_WIDTH \* 2 // 3, SCREEN\_HEIGHT \* 1 // 2)

                # Thêm bóng

                shadow\_surface = pygame.Surface((box\_rect.width + 10, box\_rect.height + 10), pygame.SRCALPHA)

                pygame.draw.rect(shadow\_surface, (0, 0, 0, 100), (5, 5, box\_rect.width, box\_rect.height), border\_radius=15)

                screen.blit(shadow\_surface, (box\_rect.x - 5, box\_rect.y - 5))

                # Vẽ khung

                pygame.draw.rect(screen, (230, 230, 230), box\_rect, border\_radius=15)

                pygame.draw.rect(screen, (50, 50, 50), box\_rect, 3, border\_radius=15)

                # Văn bản

                level\_complete\_font = pygame.font.Font(None, 48)

                text = level\_complete\_font.render("Level complete!", True, (0, 0, 0))

                text\_rect = text.get\_rect(center=(box\_rect.centerx, box\_rect.top + 80))

                screen.blit(text, text\_rect)

                # Nút

                button\_width = 150

                button\_height = 50

                button\_spacing = 10

                play\_again\_button = pygame.Rect(

                    box\_rect.centerx - button\_width // 2,

                    box\_rect.centery - button\_height - button\_spacing // 2 + 30,

                    button\_width,

                    button\_height

                )

                home\_button = pygame.Rect(

                    box\_rect.centerx - button\_width // 2,

                    box\_rect.centery + button\_spacing // 2 + 30,

                    button\_width,

                    button\_height

                )

                draw\_button(play\_again\_button, "Play Again")

                draw\_button(home\_button, "Home")

                mouse\_pos = pygame.mouse.get\_pos()

                if event.type == pygame.MOUSEBUTTONDOWN:

                    if play\_again\_button.collidepoint(mouse\_pos):

                        play\_click\_sound()

                        reset\_game()

                        square\_manager.selected\_squares = []

                        square\_manager.history = []

                        square\_manager.undo\_count = 0

                        square\_manager.score = 0

                        reset\_available = True

                        generate\_random\_squares(45)

                        game\_state = "play"

                    elif home\_button.collidepoint(mouse\_pos):

                        play\_click\_sound()

                        game\_state = "home"

        elif game\_state == "lose":

            overlay = pygame.Surface((SCREEN\_WIDTH, SCREEN\_HEIGHT), pygame.SRCALPHA)

            overlay.fill((0, 0, 0, 180))

            screen.blit(overlay, (0, 0))

            box\_rect = pygame.Rect(SCREEN\_WIDTH // 6, SCREEN\_HEIGHT // 4, SCREEN\_WIDTH \* 2 // 3, SCREEN\_HEIGHT \* 1 // 2)

            # Thêm bóng

            shadow\_surface = pygame.Surface((box\_rect.width + 10, box\_rect.height + 10), pygame.SRCALPHA)

            pygame.draw.rect(shadow\_surface, (0, 0, 0, 100), (5, 5, box\_rect.width, box\_rect.height), border\_radius=15)

            screen.blit(shadow\_surface, (box\_rect.x - 5, box\_rect.y - 5))

            # Vẽ khung

            pygame.draw.rect(screen, (230, 230, 230), box\_rect, border\_radius=15)

            pygame.draw.rect(screen, (50, 50, 50), box\_rect, 3, border\_radius=15)

            # Văn bản

            game\_over\_font = pygame.font.Font(None, 48)

            text = game\_over\_font.render("Game Over", True, (0, 0, 0))

            text\_rect = text.get\_rect(center=(box\_rect.centerx, box\_rect.top + 80))

            screen.blit(text, text\_rect)

            # Nút

            button\_width = 150

            button\_height = 50

            button\_spacing = 10

            play\_again\_button = pygame.Rect(

                box\_rect.centerx - button\_width // 2,

                box\_rect.centery - button\_height - button\_spacing // 2 + 20,

                button\_width,

                button\_height

            )

            home\_button = pygame.Rect(

                box\_rect.centerx - button\_width // 2,

                box\_rect.centery + button\_spacing // 2 + 20,

                button\_width,

                button\_height

            )

            draw\_button(play\_again\_button, "Play Again")

            draw\_button(home\_button, "Home")

            if event.type == pygame.MOUSEBUTTONDOWN:

                if play\_again\_button.collidepoint(event.pos):

                    play\_click\_sound()

                    reset\_game()

                    square\_manager.selected\_squares = []

                    square\_manager.history = []

                    square\_manager.undo\_count = 0

                    square\_manager.score = 0

                    reset\_available = True

                    generate\_random\_squares(45)

                    game\_state = "play"

                elif home\_button.collidepoint(event.pos):

                    play\_click\_sound()

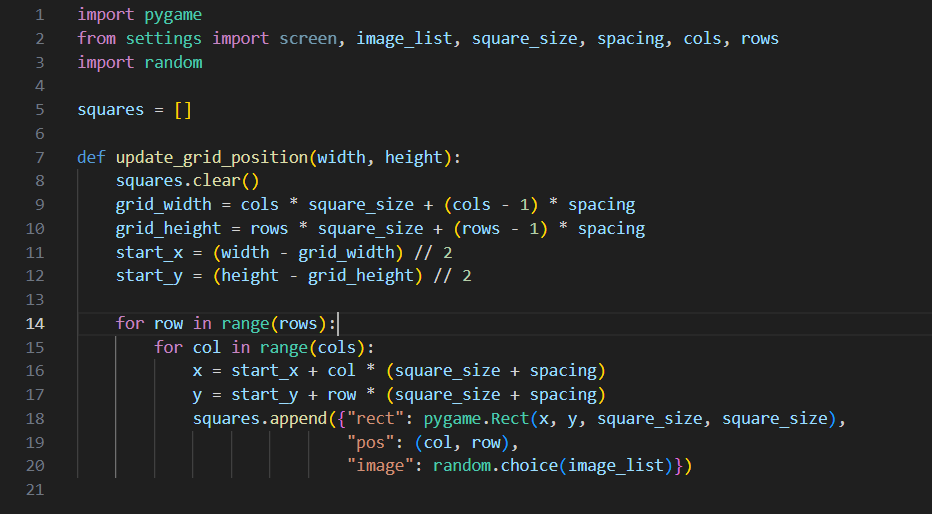
                    game\_state = "home"

        pygame.display.flip()

        pygame.time.Clock().tick(120)

    pygame.quit()

**Tiếp theo là ./player.py:**

****

**Và cuối cùng là ./settings.py:**

import pygame

import os

import pygame.freetype

import random

import sys

import json

from datetime import datetime

sys.stdout.reconfigure(encoding='utf-8')

# Khởi tạo pygame

pygame.init()

# Kích thước màn hình

SCREEN\_WIDTH, SCREEN\_HEIGHT = 500, 600

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT), pygame.RESIZABLE)

pygame.display.set\_caption("Đồ án Python")

# Màu sắc

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

GRAY = (200, 200, 200)

DARK\_GRAY = (50, 50, 50)

GREEN = (0, 128, 0)  # Màu xanh lá cho trạng thái bật âm lượng

RED = (255, 0, 0)    # Màu đỏ cho trạng thái tắt âm lượng

square\_manager = None

# Chữ

font = pygame.font.Font(None, 36)

# Định nghĩa các nút

button\_start = pygame.Rect(SCREEN\_WIDTH // 2 - 100, 250, 200, 50)

button\_music = pygame.Rect(SCREEN\_WIDTH // 2 - 100, 320, 200, 50)

# Nút trong trạng thái "Lose"

button\_retry = pygame.Rect(SCREEN\_WIDTH // 2 - 100, SCREEN\_HEIGHT // 2 + 70, 200, 50)

button\_home\_lose = pygame.Rect(SCREEN\_WIDTH // 2 - 100, SCREEN\_HEIGHT // 2 + 130, 200, 50)

# Biến trạng thái âm lượng (True: bật, False: tắt)

sound\_enabled = True

# Đường dẫn cố định

current\_dir = os.path.dirname(\_\_file\_\_)

base\_dir = os.path.abspath(os.path.join(current\_dir, '..', 'assets'))

HIGHSCORE\_FILE = os.path.join(base\_dir, "highscores.json")

MAX\_HIGHSCORES = 5

# Vẽ các nút

def draw\_button(button, text, color=GRAY):

    pygame.draw.rect(screen, color, button, border\_radius=10)

    text\_surface = font.render(text, True, BLACK)

    text\_rect = text\_surface.get\_rect(center=button.center)

    screen.blit(text\_surface, text\_rect)

# Icon và hình nền

icon = pygame.image.load(os.path.join(base\_dir, "images", "icon\_screen.jpg"))

pygame.display.set\_icon(icon)

background = pygame.image.load(os.path.join(base\_dir, "images", "icon\_screen.jpg"))

resized\_background = pygame.transform.scale(background, (SCREEN\_WIDTH, SCREEN\_HEIGHT))

# Thiết lập lưới

cols, rows = 6, 5

square\_size = 50

spacing = 5

# Âm thanh

background\_music = pygame.mixer.Sound(os.path.join(base\_dir, "sounds", "startms.mp3"))

click\_sound = pygame.mixer.Sound(os.path.join(base\_dir, "sounds", "click.mp3"))

match\_sound = pygame.mixer.Sound(os.path.join(base\_dir, "sounds", "match.mp3"))

complete\_sound = pygame.mixer.Sound(os.path.join(base\_dir, "sounds", "complete.wav"))

game\_over\_sound = pygame.mixer.Sound(os.path.join(base\_dir, "sounds", "gameover.wav"))

# Chạy âm thanh

def play\_background\_music():

    if sound\_enabled:

        pygame.mixer.Sound.play(background\_music, loops=-1)

        pygame.mixer.Sound.set\_volume(background\_music, 0.5)

def play\_click\_sound():

    if sound\_enabled:

        pygame.mixer.Sound.play(click\_sound)

        pygame.mixer.Sound.set\_volume(click\_sound, 0.7)

def play\_match\_sound():

    if sound\_enabled:

        pygame.mixer.Sound.play(match\_sound)

        pygame.mixer.Sound.set\_volume(match\_sound, 0.7)

def play\_complete\_sound():

    if sound\_enabled:

        pygame.mixer.Sound.play(complete\_sound)

        pygame.mixer.Sound.set\_volume(complete\_sound, 0.7)

def play\_gameover\_sound():

    if sound\_enabled:

        pygame.mixer.Sound.play(game\_over\_sound)

        pygame.mixer.Sound.set\_volume(game\_over\_sound, 0.7)

def stop\_background\_music():

    pygame.mixer.Sound.stop(background\_music)

# Hình ảnh cho khối

image\_list = []

image\_file = [os.path.join(base\_dir, "images", f"img{i+1}.png") for i in range(1, 5)]

for img\_f in image\_file:

    img = pygame.image.load(img\_f)

    img = pygame.transform.scale(img, (square\_size, square\_size))

    image\_list.append(img)

def reset\_game():

    from player import update\_grid\_position

    global rows, cols

    update\_grid\_position(SCREEN\_WIDTH, SCREEN\_HEIGHT)

# # Hàm để lưu high score

# def save\_highscore(score):

#     try:

#         # Đọc danh sách high score hiện có

#         if os.path.exists(HIGHSCORE\_FILE):

#             with open(HIGHSCORE\_FILE, 'r') as f:

#                 highscores = json.load(f)

#         else:

#             highscores = []

#         # Thêm score mới

#         highscores.append({

#             "score": score,

#             "date": datetime.now().strftime("%Y-%m-%d %H:%M:%S")

#         })

#         # Sắp xếp và giữ lại chỉ MAX\_HIGHSCORES bản ghi

#         highscores.sort(key=lambda x: x["score"], reverse=True)

#         highscores = highscores[:MAX\_HIGHSCORES]

#         # Lưu lại file

#         with open(HIGHSCORE\_FILE, 'w') as f:

#             json.dump(highscores, f, indent=4)

#     except Exception as e:

#         print(f"Error saving highscore: {e}")

# # Hàm để hiển thị bảng high score

# def show\_highscores():

#     try:

#         # Đọc high scores từ file

#         if os.path.exists(HIGHSCORE\_FILE):

#             with open(HIGHSCORE\_FILE, 'r') as f:

#                 highscores = json.load(f)

#         else:

#             highscores = []

#         # Tạo surface cho popup

#         popup\_width = 400

#         popup\_height = 400

#         popup = pygame.Surface((popup\_width, popup\_height), pygame.SRCALPHA)

#         popup.fill((50, 50, 50, 200))

#         pygame.draw.rect(popup, (200, 200, 200), (0, 0, popup\_width, popup\_height), border\_radius=10)

#         pygame.draw.rect(popup, (0, 0, 0), (0, 0, popup\_width, popup\_height), 2, border\_radius=10)

#         # Vẽ tiêu đề

#         title\_font = pygame.font.Font(None, 36)

#         title\_text = title\_font.render("HIGH SCORES", True, (0, 0, 0))

#         popup.blit(title\_text, (popup\_width//2 - title\_text.get\_width()//2, 20))

#         # Vẽ từng high score

#         score\_font = pygame.font.Font(None, 28)

#         if not highscores:

#             no\_scores = score\_font.render("No highscores yet!", True, (0, 0, 0))

#             popup.blit(no\_scores, (popup\_width//2 - no\_scores.get\_width()//2, 100))

#         else:

#             for i, entry in enumerate(highscores):

#                 score\_text = score\_font.render(f"{i+1}. {entry['score']} - {entry['date']}", True, (0, 0, 0))

#                 popup.blit(score\_text, (40, 80 + i\*40))

#         # Vẽ nút đóng

#         close\_button = pygame.Rect(popup\_width//2 - 50, popup\_height - 60, 100, 40)

#         pygame.draw.rect(popup, (150, 150, 150), close\_button, border\_radius=5)

#         pygame.draw.rect(popup, (0, 0, 0), close\_button, 2, border\_radius=5)

#         close\_text = score\_font.render("Close", True, (0, 0, 0))

#         popup.blit(close\_text, (close\_button.centerx - close\_text.get\_width()//2,

#                                close\_button.centery - close\_text.get\_height()//2))

#         # Hiển thị popup

#         popup\_rect = popup.get\_rect(center=(SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2))

#         screen.blit(popup, popup\_rect)

#         pygame.display.flip()

#         # Chờ người dùng nhấn nút đóng

#         waiting = True

#         while waiting:

#             for event in pygame.event.get():

#                 if event.type == pygame.QUIT:

#                     pygame.quit()

#                     exit()

#                 if event.type == pygame.MOUSEBUTTONDOWN:

#                     mouse\_pos = pygame.mouse.get\_pos()

#                     # Tính toán lại vị trí nút đóng trên màn hình

#                     screen\_close\_button = close\_button.move(popup\_rect.topleft)

#                     if screen\_close\_button.collidepoint(mouse\_pos):

#                         play\_click\_sound()

#                         waiting = False

#     except Exception as e:

#         print(f"Error showing highscores: {e}")

def home\_page():

    global sound\_enabled

    play\_background\_music()

    while True:

        screen.fill(WHITE)

        screen.blit(resized\_background, (0, 0))

        text = font.render("Tiledom", True, BLACK)

        screen.blit(text, (SCREEN\_WIDTH // 2 - text.get\_width() // 2, SCREEN\_HEIGHT // 2 - text.get\_height() // 2 - 200))

        draw\_button(button\_start, "START")

        # Vẽ nút music với trạng thái hiện tại (ON/OFF)

        music\_text = "MUSIC: ON" if sound\_enabled else "MUSIC: OFF"

        music\_color = GREEN if sound\_enabled else RED

        draw\_button(button\_music, music\_text, music\_color)

        pygame.display.flip()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                exit()

            if event.type == pygame.MOUSEBUTTONDOWN:

                if button\_start.collidepoint(event.pos):

                    play\_click\_sound()

                    print("Đã nhấn vào bắt đầu")

                    reset\_game()

                    game\_state = "play"

                    return

                if button\_music.collidepoint(event.pos):

                    play\_click\_sound()

                    # Bật/tắt âm thanh

                    sound\_enabled = not sound\_enabled

                    if sound\_enabled:

                        play\_background\_music()

                    else:

                        stop\_background\_music()

def lose():

    global game\_state

    game\_state = "lose"

    play\_gameover\_sound()

    #save\_highscore(square\_manager.score)  # Lưu điểm khi thua

    box\_rect = pygame.Rect(SCREEN\_WIDTH // 4, SCREEN\_HEIGHT // 3, SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 3)

    while True:

        overlay = pygame.Surface((SCREEN\_WIDTH, SCREEN\_HEIGHT), pygame.SRCALPHA)

        overlay.fill((0, 0, 0, 180))

        screen.blit(overlay, (0, 0))

        pygame.draw.rect(screen, (255, 255, 255), box\_rect, border\_radius=10)

        pygame.draw.rect(screen, (0, 0, 0), box\_rect, 3, border\_radius=10)

        text = font.render("Game Over", True, (0, 0, 0))

        screen.blit(text, (SCREEN\_WIDTH // 2 - text.get\_width() // 2, SCREEN\_HEIGHT // 2 - 20))

        draw\_button(button\_retry, "Chơi lại")

        draw\_button(button\_home\_lose, "Trang chủ")

        pygame.display.flip()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                exit()

            if event.type == pygame.MOUSEBUTTONDOWN:

                if button\_retry.collidepoint(event.pos):

                    play\_click\_sound()

                    reset\_game()

                    return "play"

                if button\_home\_lose.collidepoint(event.pos):

                    play\_click\_sound()

                    return "home"