

UỶ BAN NHÂN DÂN  
THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC SÀI GÒN



BÁO CÁO TIỂU LUẬN  
NGÔN NGỮ LẬP TRÌNH PYTHON

Tên đề tài: Xây dựng game bắn súng 2D

Oak Hunter

Thông tin thành viên:

Nguyễn Nam Dương – 3123560012

Lớp: DKP1231

Giảng viên: Nguyễn Trung Tín

Thành phố Hồ Chí Minh, ngày 04 tháng 12 năm 2024



## **NHẬN XÉT VÀ 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 của trường 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 đặc biệt cảm ơn thầy Nguyễn Trung Tín, thầy là giảng viên giảng dạy đã nhiệt tình hướng dẫn giúp em hoàn thành được 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 sót, em rất mong nhận được ý kiến đóng góp của thầy, cô để em học hỏi được nhiều kĩ năng, kinh nghiệm và sẽ hoàn thành tốt hơn cho những bài báo cáo sắp tới.

Em xin chân thành cảm ơn

# MỤC LỤC

<b>PHẦN I</b> .....	<b>8</b>
1. Giới thiệu đề tài.....	8
2. Lý do chọn đề tài.....	8
3. Mục đích và mục tiêu của đề tài .....	9
<b>PHẦN II</b> .....	<b>10</b>
1. Đôi nét về game Oak Hunter và nguồn cảm hứng.....	10
1.1 Quá trình ra đời và ý tưởng của Oak Hunter .....	10
1.2 Mục tiêu của game Oak Hunter .....	10
2. Mục tiêu của đồ án.....	10
3. Tiến hành xây dựng ứng dụng.....	10
3.1 Cài đặt các hằng số để sử dụng nhiều lần trong trò chơi .....	11
3.2 Lớp Button (button.py) .....	12
3.3 Lớp Enemy (Enemy.py).....	13
3.4 Cài đặt thư viện, khởi tạo các biến trong cửa sổ trò chơi.....	24
3.5 Sử dụng các assets cho hình ảnh, âm thanh và các nút ở trong cửa sổ trò chơi.....	27
3.6 Tạo các nhóm sprite có trong trò chơi .....	29
3.7 Tổ chức tạo các lớp có liên quan tới người chơi.....	30
3.8 Lớp đạn dược (Bullet class) .....	39
3.9 Lớp Đồng xu (CoinBar class và Coin class).....	40
3.10 Lớp trang trí (Decoration class) .....	42
3.11 Lớp Bẫy (Trap class) .....	43
3.12 Lớp cấp độ kế tiếp (NextLevel class).....	45
3.13 Lớp thế giới (World class).....	46
3.14 Khởi tạo các mảng lưu trữ các thành phần của thế giới .....	49
3.15 Các hàm hỗ trợ trong trò chơi .....	51
3.16 Hàm xây dựng cửa sổ trò chơi.....	55
3.17 Một vài hình ảnh của trò chơi.....	61
4. Kết thúc và kết luận.....	63
<b>Phần III</b> .....	<b>64</b>



# PHẦN I

## MỞ ĐẦU

### 1. Giới thiệu đề tài

Tên đề tài của nhóm:

Xây dựng trò chơi bắn súng Oak Hunter, một trò chơi bắn súng 2D vượt ải bằng ngôn ngữ Python và thư viện hỗ trợ Pygame.

### 2. Lý do chọn đề tài

Những năm gần đây, khi hạ tầng công nghệ thông tin được mở rộng đầu tư, các địa phương trong cả nước đã đẩy mạnh ứng dụng công nghệ thông tin, gắn kết chặt chẽ với thúc đẩy cải cách hành chính, xây dựng chính phủ điện tử để phục vụ tốt hơn nhu cầu của người dân và doanh nghiệp. Đến nay, tất cả các bộ, ngành, địa phương đều đã có trang thông tin điện tử, cung cấp thông tin, dịch vụ công trực tuyến cho người dân và doanh nghiệp, góp phần giảm thời gian, chi phí thủ tục hành chính, tăng tính minh bạch, hiệu quả. Trong các lĩnh vực quan trọng như thuế, hải quan, bảo hiểm xã hội... việc cải cách thủ tục hành chính được chú trọng... Với ngành giáo dục, việc ứng dụng **CÔNG NGHỆ THÔNG TIN** được phổ cập tại hầu hết các trường trung học phổ thông và gần 80% các trường đại học, cao đẳng trên toàn quốc.

Python là một trong những ngôn ngữ lập trình được sử dụng phổ biến nhất hiện nay. Thích hợp cho người mới bắt đầu bởi vì ngôn ngữ dễ học. Nó là một ngôn ngữ lập trình open-source miễn phí với các module hỗ trợ mở rộng và phát triển cộng đồng, dễ dàng tích hợp với các dịch vụ web, cấu trúc dữ liệu thân thiện với user và GUI-based desktop app. Nó là một ngôn ngữ lập trình phổ biến cho các ứng dụng machine learning và deep learning.

Trong quá trình tìm hiểu em thấy rất hứng thú với các ứng dụng game được cài đặt và lập trình bằng ngôn ngữ lập trình Python bằng thư viện Pygame. Pygame là một bộ

mô-đun Python đa nền tảng được thiết kế để viết trò chơi điện tử. Nó bao gồm đồ họa máy tính và thư viện âm thanh được thiết kế để sử dụng với ngôn ngữ lập trình Python và cũng nhờ nó mà có rất nhiều tựa game được viết bằng ngôn ngữ Python đã ra đời. Em đã quyết định sử dụng Python cùng với thư viện Pygame để xây dựng 1 tựa game bắn súng 2D vượt ải, đặt tên là **Oak Hunter**, được lấy cảm hứng từ tựa game bắn súng 2D 1 thời lùng lẫy ở Việt Nam trên hệ máy NES, Contra.

### 3. Mục đích và mục tiêu của đề tài

- Mục đích của đề tài:

- Nắm chắc được kỹ năng và kiến thức về lập trình
- Tìm hiểu về thư viện Python trong ngôn ngữ lập trình Python
- Củng cố, áp dụng, nâng cao kiến thức đã học
- Nắm bắt được quy trình làm game cơ bản

- Mục tiêu của đề bài:

- Vận dụng được tính chất của lập trình hướng đối tượng
- Sử dụng thư viện Pygame vào việc xây dựng game bắn súng 2D

## **PHẦN II**

### **NỘI DUNG**

#### **1. Đôi nét về game Oak Hunter và nguồn cảm hứng**

##### **1.1 Quá trình ra đời và ý tưởng của Oak Hunter**

Nguồn cảm hứng của em đến từ tựa game Contra huyền thoại, một thể loại game Run&Gun đổ bộ vào Việt Nam nhưng năm đầu thập niên 90. Contra là một trong những tựa game hành động bắn súng kinh điển được phát triển bởi Konami. Ra mắt lần đầu vào năm 1987, Contra nhanh chóng trở thành một biểu tượng của thể loại Gun&Run. Người chơi sẽ được vào vai các nhân vật Bill và Lance, họ chiến đấu với đội quân kẻ thù đến từ không gian ngoài hành tinh. Với lối hơi nhanh nhẹn, đồ họa hấp dẫn cùng với âm nhạc đặc sắc, Contra đã ghi điểm với cộng đồng game thủ trên toàn thế giới suốt nhiều thập kỷ.

Đó cũng chính là lý do em quyết định tạo ra tựa game có thể loại là Gun&Run và lấy bối cảnh là chiến đấu chống lại quái vật ở rừng thông nên em quyết định đặt tên cho game là Oak Hunter.

##### **1.2 Mục tiêu của game Oak Hunter**

Mục tiêu của người chơi trong trò chơi là điều khiển nhân vật chính của chúng ta, Gangster sẽ vượt qua những kẻ địch nguy hiểm, chông gai, những góc di chuyển với mục đích cuối cùng là chiến thắng được tên trùm Golem để ngăn chặn được kế hoạch thống trị thế giới của hắn bằng đội quân quái vật

#### **2. Mục tiêu của đồ án**

Xây dựng 1 ứng dụng trò chơi giống với thể loại Gun&Run của tựa game Contra dựa vào kiến thức đã học trong suốt quá trình học về ngôn ngữ lập trình Python cùng với sự hỗ trợ của các thư viện của Python

#### **3. Tiến hành xây dựng ứng dụng**

**Sơ lược:** cấu trúc của chương trình được phân thành 4 file chính, 1 Folder “Level” dùng để lưu trữ các file csv (nhằm để tạo màn trò chơi) các Folder còn lại có nhiệm vụ lưu trữ ảnh và âm thanh trò chơi.

+ File settings.py: để lưu trữ các biến cố định(hàng số) được sử dụng xuyên suốt trong trò chơi (được đề cập ở mục 3.1)

+ File button.py: dùng để tạo nút cho trò chơi, sở dĩ được tạo riêng nhằm để dễ dàng tái sử dụng phương thức tạo nút (được đề cập ở mục 3.2)

+ File Enemy.py: là các module các loại quái vật có trong game (được đề cập ở mục 3.3)

+ File main.py: là file chính để khởi động trò chơi (được đề cập ở mục 3.4 đến 3.17)

📁 __pycache__	12/4/2024 8:44 AM	File folder
📁 Audio	12/3/2024 9:50 AM	File folder
📁 Entity	12/4/2024 8:38 AM	File folder
📁 Level	12/3/2024 8:02 AM	File folder
📁 MainGame	12/3/2024 8:02 AM	File folder
📁 Map	12/3/2024 8:02 AM	File folder
button.py	12/4/2024 7:32 AM	Python Source File 1 KB
Enemy.py	12/4/2024 8:44 AM	Python Source File 10 KB
main.py	12/4/2024 8:40 AM	Python Source File 53 KB
settings.py	12/4/2024 8:29 AM	Python Source File 1 KB

### 3.1 Cài đặt các hằng số để sử dụng nhiều lần trong trò chơi

Các biến định nghĩa cho cửa sổ trò chơi, số lượng vật thể có trong trò chơi

```
python settings.py > ...
1 import os
2
3 SCREEN_WIDTH = 1024
4 SCREEN_HEIGHT = 512
5 WIDTH_MAP = 704
6 HEIGHT_MAP = 416
7 FPS = 60
8 ANIMATION = 100
9 GRAVITY = 0.75
10 SCROLL_THRESH = 400
11 ROWS = 16
12 COLS = 150
13 TILE_SIZE = SCREEN_HEIGHT // ROWS # Này dùng trong game
14 TILE_SIZE_MAP = HEIGHT_MAP // ROWS # Này dùng trong map
15 TILE_TYPES = sum(1 for file in os.listdir('Map/Tile') if file.endswith('.png'))
16 MAX_LEVELS = 5
17 MAINBTN_WIDTH = 150
18 MAINBTN_HEIGHT = 66
```

- **SCREEN\_WIDTH** và **SCREEN\_HEIGHT**: để định nghĩa kích thước của màn hình chính trò chơi còn **WIDTH\_MAP** và **HEIGHT\_MAP** định nghĩa kích thước của màn hình phụ trò chơi, **TILE\_SIZE** là định nghĩa độ rộng và độ cao của mỗi ô trong cửa sổ chính của trò chơi còn **TILE\_SIZE\_MAP** là định nghĩa độ rộng và độ cao mỗi ô khi dùng hàm “map\_editor” (cho phép tham gia chỉnh sửa bản đồ của trò chơi).

- **GRAVITY** là hằng số về trọng lực, **ANIMATION** là hằng số giúp trong việc chuyển đổi các ảnh của các nhân vật cũng như quái vật được ổn định và mượt mà, **SCROLL\_THRESH** là ngưỡng cuộn màn hình. **ROWS** và **COLS** là số lượng loại ô trong bản đồ là 150 cột và 16 dòng và cấp độ tối đa cho game là **MAX\_LEVELS**. **MAINBTN\_WIDTH**, **MAINBTN\_HEIGHT** là chiều dài và chiều dọc của các nút ở của trang chủ của trò chơi

## Định nghĩa mã màu và chiều rộng, chiều cao của vật thể

```
20     PLAYER_WIDTH = 128
21     PLAYER_HEIGHT = 128
22
23     SHOP_WIDTH = 118
24     SHOP_HEIGHT = 128
25
26     GOLEM_WIDTH = 100
27     GOLEM_HEIGHT = 100
28
29     COIN_WIDTH = 16
30     COIN_HEIGHT = 16
31
32     RED = (255, 0, 0)
33     WHITE = (255, 255, 255)
34     BLACK = (0, 0, 0)
35     GRAY = (100, 100, 100)
```

- Các mã màu được định nghĩa bằng mã màu RGB: **RED**, **WHITE**, **BLACK** và **GRAY**
- Các biến còn lại được định nghĩa theo cặp chiều rộng và chiều cao của nhân vật (**PLAYER\_WIDTH** và **PLAYER\_HEIGHT**), (**SHOP\_WIDTH** và **SHOP\_HEIGHT**) là cửa cửa hàng, (**COIN\_WIDTH** và **COIN\_HEIGHT**) là cửa đồng xu

### 3.2 Lớp Button (button.py)

Định nghĩa lớp Button với các thuộc tính

- **x, y:** là tạo độ của nút trên mà hình

- **image, image\_hover:** là hình ảnh ban đầu và hình ảnh khi di chuyển vào nút của trò chơi
- **scale:** tỷ lệ thay đổi kích thước hình ảnh của nút
- **self.image:** là hình ảnh ban đầu của trò chơi khi chưa hover
- **self.image\_hover:** là hình ảnh khi di chuyển chuột vào của nút
- **self.rect:** vùng giới hạn của hình ảnh nút
- **self.clicked:** là biến boolean để kiểm tra nút ấn vào chưa

Phương thức **draw()** được sử dụng để vẽ nút lên màn hình và kiểm tra xem nút có được ấn vào không. Cụ thể như sau:

- **pos = pygame.mouse.get\_pos():** lấy vị trí của chuột.
- **if self.rect.collidepoint(pos):** kiểm tra xem chuột có ở trên nút không.
- **if pygame.mouse.get\_pressed()[0] == 1 and not self.clicked:** kiểm tra xem chuột có được nhán không và nút chưa được nhán trước đó.
- **action = True:** đánh dấu rằng nút đã được nhán.
- **if pygame.mouse.get\_pressed()[0] == 0::** kiểm tra xem chuột có được nhả ra không, nếu có thì đặt **self.clicked = False** để cho phép nhấn nút tiếp theo.
- **surface.blit(self.image, (self.rect.x, self.rect.y)):** vẽ hình ảnh nút lên màn hình.

- **return action:** trả về **True** nếu nút được nhấn và **False** nếu không

```
button.py > ...
1 import pygame
2
3 class Button():
4     def __init__(self,x, y, image, image_hover, scale):
5         self.image = pygame.transform.scale(image, (int(image.get_width() * scale), int(image.get_height() * scale)))
6         self.image_hover = pygame.transform.scale(image_hover, (int(image.get_width() * scale), int(image.get_height() * scale)))
7         self.rect = self.image.get_rect()
8         self.rect.topleft = (x, y)
9         self.clicked = False
10
11    def draw(self, surface):
12        action = False
13        surface.blit(self.image, (self.rect.x, self.rect.y))
14        #get mouse position
15        pos = pygame.mouse.get_pos()
16        # Kiểm tra chuột có ăn hay di chuyển vào khu vực không để đổi hình ảnh
17        if self.rect.collidepoint(pos):
18            if self.image_hover:
19                surface.blit(self.image_hover, (self.rect.x, self.rect.y))
20            if pygame.mouse.get_pressed()[0] == 1 and self.clicked == False:
21                action = True
22                self.clicked = True
23            if pygame.mouse.get_pressed()[0] == 0:
24                self.clicked = False
25        return action
26
```

### 3.3 Lớp Enemy (Enemy.py)

#### Cài đặt hình ảnh và các loại hành động của Enemy

```
6 class Enemy(pygame.sprite.Sprite):
7     def __init__(self, type, x, y, scale):
8         pygame.sprite.Sprite.__init__(self)
9         # Cài đặt hình ảnh cho quái vật
10        self.type = type
11        animation_types = ['Idle', 'Walk', 'Attack', 'Die', 'Hurt']
12        folder_path = f'Entity/{self.type}'
13        self.update_time = pygame.time.get_ticks()
14        self.action = 0
15        self.frame_index = 0
16        self.animation_list = []
```

1. **pygame.sprite.Sprite.\_\_init\_\_(self):** khởi tạo lớp quái vật đặt tên là Enem, lớp lớp **pygame.sprite.Sprite**
2. **self.type:** là loại quái vật, 1 trong 4 loại 'Skeleton', 'Bigger', 'Demon' và 'Boss'
3. **animation\_type:** là các loại hành động của các quái vật thuộc lớp Enemy đó
4. **folder\_path:** là đường dẫn để dẫn đến folder chứa các frame về từng loại hành động của quái vật
5. **self.update\_time:** dùng để lưu trữ hoặc quản lý thông tin về thời gian cập nhật của hình ảnh
6. **self.action:** để đánh dấu hành động thứ mấy trong 'animation\_types' của quái vật
7. **self.frame\_index:** được dùng để trả tới từng hành động trong 'self.action'
8. **self.animation\_list:** là danh sách 2 chiều chứ từng hành động, và từng hành động đó lại chứ các frame ảnh tạo ra hiệu ứng chuyển động 2D

**Tiếp theo đó là làm hoạt ảnh cho quái vật. Gồm có các trạng thái 'Idle', 'Walk', 'Hurt' và 'Die' được lưu trong từng folder tương ứng với tên trạng thái đó**

Name	Date modified	Type	Size
Attack	12/3/2024 8:02 AM	File folder	
Die	12/3/2024 8:02 AM	File folder	
Hurt	12/3/2024 8:02 AM	File folder	
Idle	12/3/2024 8:02 AM	File folder	
Walk	12/3/2024 8:02 AM	File folder	

### Vòng for xử lý đưa hình ảnh trạng thái vào danh sách

```

17     for animation in animation_types:
18         temp_list = []
19         number_frames = len(os.listdir(f'{folder_path}/{animation}'))
20         for i in range(1, number_frames + 1):
21             frames = pygame.image.load(f'{folder_path}/{animation}/{i}.png').convert_alpha()
22             frames = pygame.transform.scale(frames, (int(frames.get_width() * scale), int(frames.get_height() * scale)))
23             temp_list.append(frames)
24         self.animation_list.append(temp_list)
25     self.image = self.animation_list[self.action][self.frame_index]
26     self.rect = self.image.get_rect()
27     self.rect.center = (x, y)
28     self.width = self.rect.width()
29     self.height = self.rect.height()

```

1. Vòng lặp **for** đầu tiên (**for animation in animation\_types**) lặp qua từng loại animation
  - **temp\_list**: là một danh sách tạm thời để lưu trữ các hình ảnh của animation hiện tại, đầu tiên khai báo cho nó là rỗng
  - **number\_frames**: là lấy số lượng hình ảnh có trong thư mục animation tương ứng
  - Vòng lặp for thứ 2 (**for i in range(1, number\_frames + 1)**): lặp qua tất cả các hình ảnh trong thư mục. Bắt đầu là vị trí 1 (VD: 1.png)
    - Hình ảnh được tải lên từ tệp tin tương ứng và chuyển đổi kích thước sao cho phù hợp với quy mô đã cho (**scale**)
    - Hình ảnh sau khi được tải và chuyển đổi kích thước sẽ được thêm vào **temp\_list**



**Ví dụ:** tải ảnh lên và thêm vào sách hoạt ảnh, ở đây số lượng ảnh trong thư mục Die tương ứng là 22 ảnh

2. **self.animation\_list.append(temp\_list)**: sau khi lặp qua tất cả hình ảnh trong thư mục animation, **temp\_list** sẽ được thêm vào **self.animation\_list**. Điều này giúp tạo ra một danh sách 2D chứa tất cả các hình ảnh của mọi loại animtion cho quái vật
3. **self.image = self.animation\_list[self.action][self.frame\_index]**: Hình ảnh hiện tại của nhân vật được thiết lập bằng cách chọn một hình ảnh từ danh sách animation tương ứng với hành động hiện tại (**self.action**) và chỉ số khung hình hiện tại (**self.frame\_index**)
4. **self.rect**: vị trí và kích thước của hình ảnh nhân vật được cập nhật dựa trên hình ảnh hiện tại
5. **self.width = self.image.get\_width()**: lấy chiều rộng của hình ảnh
6. **self.height = self.image.get\_height()**: lấy chiều cao của hình ảnh

## Cài đặt biến cho lớp Enemy

### 1. Cài đặt tầm nhìn tương ứng với từng loại

```

31     if self.type == 'Skeleton':
32         self.health = 100
33         self.dame = 10
34         self.vision = pygame.Rect(0, 0, 100, self.height)
35         self.vision_width = 100
36     elif self.type == 'Bigger':
37         self.health = 150
38         self.dame = 20
39         self.vision = pygame.Rect(0, 0, 200, self.height)
40         self.vision_width = 200
41     elif self.type == 'Demon':
42         self.health = 500
43         self.dame = 50
44         self.vision_width = 200
45         self.vision = pygame.Rect(0, 0, 300, self.height)
46         self.vision_width = 300
47     elif self.type == 'Boss':
48         self.health = 1000
49         self.dame = 100
50         self.vision = pygame.Rect(0, 0, 400, self.height)
51         self.vision_width = 400

```

Cài đặt các biến ‘self.health’, ‘self.dame’, ‘self.vision’, ‘self.vision\_width’ dựa trên biến ‘self.type’ tức là loại quái vật

- **self.health**: cài đặt máu cho từng loại quái vật
- **self.dame**: cài đặt lượng dame tương ứng với loại quái vật
- **self.vision**: cài đặt tầm nhìn của quái vật, là loại hình chữ nhật (pygame.Rect)
- **self.vision\_width**: là chiều rộng của tầm nhìn tương ứng với từng loại

### 2. Khai báo các biến cần thiết để thực thi hành động

```

52     self.speed = 1
53     self.direction = 1
54     self.flip = False
55     # Các biến dùng để di chuyển
56     self.moving_left = False
57     self.moving_right = False
58     self.attack = False
59     self.hurt = False
60     self.vel_y = 0
61     # Biến chờ đợi là hành phúc
62     self.dame_coldown = 0
63     # Biến dành cho ai
64     self.move_counter = 0
65     self.idling = False
66     self.idling_counter = 0
67     # Biến kiểm tra va chạm
68     self.collision_rect = pygame.Rect(self.rect.centerx - 5 * scale, self.rect.centery - 10 * scale, 10 * scale, 42 * scale)
69

```

- **self.speed = 1**: là tốc độ di chuyển của quái vật bằng 1
- **self.direction = 1**: là hướng đi của quái vật. Mặc định là 1 (hướng phải) còn ngược lại là -1 (hướng trái)
- **self.flip = False**: biến boolean để xác định hướng nhìn của quái vật (True là hướng trái còn False là hướng Phải)
- **self.moving\_left = False**: biến boolean để giúp quái vật có thể di chuyển sang bên trái
- **self.moving\_right = False**: biến boolean để giúp quái vật có thể di chuyển sang bên phải
- **self.attack = False**: biến boolean để kiểm tra quái vật có tấn công hay không
- **self.hurt = False**: biến boolean để kiểm tra quái vật có thực thi hành động dính sát thương (hurt) hay không
- **self.move\_counter**: được dùng cho việc đếm số bước di chuyển
- **self.idling = False**: biến boolean để xác định xem có đang ở trạng thái idle hay không
- **self.idling\_counter = 0**: Biến đếm để xác định thời gian ‘Idle’(nhàn rỗi)
- **self.collision\_rect = pygame.Rect(self.rect.centerx - 5 \* scale, self.rect.centery - 10 \* scale, 10 \* scale, 42 \* scale)**: là vị trí va chạm của quái vật

```

70     def update_action(self, new_action):
71         if new_action != self.action:
72             self.action = new_action
73             self.frame_index = 0
74

```

Phương thức **update\_action()** này được gọi để khi chuyển hành động này sang hành động khác. Khi chuyển đổi sang hành động mới sẽ reset lại biến ‘self.frame\_index’ lại bằng 0

```

def update_animation(self):
    self.image = self.animation_list[self.action][self.frame_index]
    if pygame.time.get_ticks() - self.update_time > ANIMATION:
        self.update_time = pygame.time.get_ticks()
        self.frame_index += 1
    if self.frame_index >= len(self.animation_list[self.action]):
        if self.action == 3: # Die
            self.frame_index = len(self.animation_list[self.action]) - 1
        elif self.action == 4: # Hurt
            self.update_action(0)
            self.hurt = False
        elif self.action == 2: # Attack
            self.update_action(0)
            self.attack = False
        else:
            self.frame_index = 0

```

Phương thức **update\_animation()** này là phương thức giúp cập nhật hoạt ảnh

### 1. Cập nhật hình ảnh của nhân vật:

Hình ảnh của quái vật sẽ được cập nhật bằng cách lấy từ **self.animation\_list** tương ứng với hành động (**self.action**) và chỉ số frame hiện tại (**self.frame\_index**)

### 2. Kiểm tra thời gian cập nhật animation:

Nếu đủ thời gian kể từ lần cập nhật trước đó (đã vượt qua thời gian cooldown **ANIMTION**) thì **self.update\_time** được cập nhật với thời điểm hiện tại, chỉ số **self.frame\_index** được tăng lên 1

### 3. Kiểm tra kết thúc của animation:

Nếu chỉ số **self.frame\_index** vượt quá số lượng frame trong animtion hiện tại, nghĩa là animtion đã kết thúc

- Nếu hành động hiện tại là ‘Die’ (**self.action == 3**), chỉ số frame được đặt lại là frame cuối cùng của animation
- Nếu hành động hiện tại là ‘Hurt’ hoặc ‘Attack’ thì cập nhật lại hành động nhàn rỗi ‘Idle’
- Ngược lại nếu không phải 3 hành động trên, tức là hành động ‘Idle’ thì frame đặt lại là 0, bắt đầu animation từ đầu

```

92     def move(self, world):
93         dx = 0
94         dy = 0
95
96         # Điều chỉnh theo hướng di chuyển
97         if self.moving_right:
98             dx = self.speed
99             self.direction = 1
100            self.flip = False
101        if self.moving_left:
102            dx = -self.speed
103            self.direction = -1
104            self.flip = True
105        # Thêm trọng lực
106        self.vel_y += GRAVITY
107        if self.vel_y > 10:
108            self.vel_y = 10
109        dy += self.vel_y
110

```

Phương thức **move()** là phương thức di chuyển cho quái vật

### 1. Khởi tạo biến

**dx, dy:** là các biến di chuyển theo trục x và trục y

### 2. Xác định hướng di chuyển

- Nếu **self.moving\_left** là True, nhân vật di chuyển sang trái, **dx** sẽ được gán là âm, **self.flip** sẽ được đặt thành True và **self.direction** được đặt thành -1, ở đây là đi về bên trái
- Tương tự, nếu **self.moving\_right** là True, nhân vật di chuyển sang phải, **dx** được gán là dương, **self.flip** được đặt thành False và **self.direction** được đặt thành 1, nhân vật sẽ đi về hướng bên phải

### 3. Áp dụng trọng lực

- Vận tốc trục y của nhân vật sẽ được cộng thêm giá trị bằng hằng số trọng lực **GRAVITY**
- Nếu vận tốc trục y lớn hơn 10 (mức tối đa), nó sẽ không tăng thêm
- Làm như thế để tạo cảm giác trọng lực thực tế, càng rơi lâu xuống vận tốc càng nhanh
- Vị trí **dy** của nhân vật sẽ được cập nhật bằng cách thêm vận tốc trên trục y **self.vel\_y** vào **dy**

```

111     # Kiểm tra va chạm
112     for tile in world.obstacle_list:
113         # Trục x
114         if tile[1].colliderect(self.collision_rect.x + dx, self.collision_rect.y, self.collision_rect.width, self.collision_rect.height):
115             dx = 0
116             self.direction *= -1
117             self.move_counter += 1
118         # Trục y
119         if tile[1].colliderect(self.collision_rect.x, self.collision_rect.y + dy, self.collision_rect.width, self.collision_rect.height):
120             if self.vel_y < 0:
121                 self.vel_y = 0
122                 dy = tile[1].bottom - self.collision_rect.top
123             elif self.vel_y >= 0:
124                 self.vel_y = 0
125                 self.in_air = False
126                 dy = tile[1].top - self.collision_rect.bottom
127
128         if self.rect.top > SCREEN_HEIGHT:
129             self.health = 0
130             dy = 0
131         # Cập nhật vị trí của collision_rect
132         self.collision_rect.x += dx
133         self.collision_rect.y += dy
134
135         # Cập nhật vị trí của self.rect để đồng bộ hóa với collision_rect
136         self.rect.centerx = self.collision_rect.centerx
137         self.rect.bottom = self.collision_rect.bottom

```

Phần tiếp theo của phương thức **move()** là xử lý va chạm giữa quái vật với các vật khác, cụ thể là bề mặt của các chướng ngại vật có trong ‘world’

### 1. Kiểm tra va chạm

- Vòng lặp **for** duyệt qua danh sách **world.obstacle\_list**, kiểm tra va chạm giữa các nhân vật và lớp vật cản
- Kiểm tra va chạm theo hướng x:
  - Nếu có va chạm theo hướng x, **dx** sẽ được đặt về 0, ngăn cho quái vật di chuyển tiếp tục theo hướng x
  - Nếu quái vật vô tình chạm vào bề mặt của vật cản theo hướng x thì đảo chiều hướng đi của quái vật bằng cách nhân biến **self.direction** cho -1 và tăng biến **self.move\_counter** lên 1, đây chính là cách mà quái vật thông thường di chuyển
- Kiểm tra va chạm theo hướng y:
  - Nếu quái vật đang ở trên không tức là biến **self.vel\_y >= 0** thì biến **self.vel\_y** được đặt về 0, và **dy** được điều chỉnh làm sao cho nhân vật đứng trên mặt đất

### 2. Kiểm tra va chạm với phần dưới màn hình:

- Sử dụng điều kiện kiểm tra **self.rect.top** nếu lớn hơn chiều rộng của màn hình **SCREEN\_HEIGHT**
- Nếu điều kiện đúng thì ngay lập tức cho máu quái vật về bằng 0 và cho máu về bằng 0

### 3. Cập nhật vị trí:

- Cập nhật vị trí va chạm
  - Cập nhật vị trí của **self.collision\_rect.x** tức là vị trí va chạm của trục x thêm với giá trị **dx**

- Cập nhật vị trí của **self.collision\_rect.y** tức là vị trí va chạm của trục y thêm với giá trị **dy**
- Cập nhật vị trí hình ảnh
  - Cập nhật vị trí của **self.rect.centerx** tức là vị trí của hình ảnh vị trí ở giữa theo trục x bằng với vị trí ở giữa của vị trí va chạm **self.collision\_rect.centerx**
  - Cập nhật vị trí của **self.rect.bottom** tức là vị trí của hình ảnh ở vị toạ độ phía dưới hình ảnh bằng với vị trí va chạm ở toạ độ phía dưới hình ảnh **self.collision\_rect.bottom**

```

197     def update(self):
198         # ['Idle', 'Walk', 'Attack', 'Die', 'Hurt']
199         if self.health <= 0:
200             self.health = 0
201             self.update_action(3)
202             self.hurt = False
203             self.moving_left = False
204             self.moving_right = False
205             self.attack = False
206         else:
207             if self.moving_left or self.moving_right:
208                 self.update_action(1)
209             elif self.attack:
210                 self.update_action(2)
211             elif self.hurt:
212                 self.update_action(4)
213             else:
214                 self.update_action(0)
215         if self.dame_cooldown > 0:
216             self.dame_cooldown -= 1

```

Tiếp đến là phương thức **update()** có nhiệm vụ là cập nhật lại hành động của kẻ địch cho đúng với trạng thái hiện tại, đồng thời cập nhật lại các biến cooldown

## 1. Cập nhật trạng thái hành động

- Ở câu điều kiện đầu tiên **if self.health <= 0** tức là kiểm tra máu hiện tại nếu nhỏ hơn hoặc bằng 0 thì ngay lập tức cho máu hiện tại bằng 0, **self.health = 0** và cập nhật hành động bằng cách gọi lại phương thức **update\_action()** và truyền cho nó tham số là **3**, tại hành động 'Die'. Tiếp đến cập nhật toàn bộ biến dùng để kiểm tra đi lại, tấn công và nhận sát thương thành **False**
- Ở câu điều kiện ngược lại, **else** kiểm tra nếu hiện tại quái vật đang di chuyển sang trái **self.moving\_left** hoặc quái vật đang di chuyển sang phải **self.moving\_right** thì cập nhật hành động di chuyển, nếu đang tấn công **self.attack** thì cập nhật hành động tấn công và nếu đang nhận sát thương **self.hurt** thì cập nhật hành động nhận sát thương bằng cách sử dụng phương thức **upadte\_action()** và truyền các tham số lần lượt là **1, 2, 4** còn nếu đang nghỉ ngơi thì truyền tham số **0**

## 2. Cập nhật lại biến cooldown

**if self.dame\_cooldown > 0** biến **self.dame\_cooldown** là biến tính thời gian gây sát thương lên người chơi. Nếu biến này lớn hơn 0 thì sẽ được giảm đi 1 mỗi lần gọi hàm **update()**

Tiếp theo là phương thức di chuyển và hành động của quái vật cũng là hàm quan trọng nhất của mỗi loại quái vật, phương thức này em đặt tên là **ai()**

```
138     def ai(self, player, screen_scroll, world):
139         # Cập nhật tầm nhìn
140         if self.direction == -1:
141             self.vision.x = self.rect.centerx + self.vision_width * self.direction
142         else:
143             self.vision.x = self.rect.centerx
144             self.vision.y = self.rect.y
145         if self.health > 0:
146             if self.vision.colliderect(player.rect) and player.health > 0:
147                 self.idling = True
148                 self.idling_counter = 10
149                 self.speed = 2
150                 if abs(self.rect.centerx - player.rect.centerx) > 30:
151                     if self.rect.centerx > player.rect.centerx:
152                         self.moving_left = True
153                         self.moving_right = False
154                     else:
155                         self.moving_left = False
156                         self.moving_right = True
157                     self.move(world)
158                 else:
159                     if abs(self.rect.centery - player.rect.centery) <= 30:
160                         self.moving_left = False
161                         self.moving_right = False
162                         self.attack = True
163                         if self.attack == True:
164                             if self.dame_cooldown == 0:
165                                 player.health -= self.dame
166                                 self.dame_cooldown = 100
167                                 if random.randint(1, 3) == 1:
168                                     player.hurt = True
169                         else:
170                             self.speed = 1
```

## 1. Cập nhật lại tầm nhìn của quái vật

Kiểm tra hướng đi của quái vật

- Nếu hướng đi của quái vật **if self.direction == -1** tức là quái vật đang có hướng đi về bên trái thì tầm nhìn sẽ được cập nhật trục x **self.vision.x = self.rect.centerx** cộng với chiều dài của tầm nhìn **self.vision\_width** và nhân với hướng đi hiện tại
- Ngược lại nếu đang có tầm nhìn hiện tại là 1 thì chỉ cập nhật lại trục x sao cho bằng với trục x của kích thước hình ảnh của quái vật **self.rect.x**



Minh họa tầm nhìn của quái vật

## 2. Kiểm tra sự sống của quái vật và cập nhật tấn công

Nếu quái vật đang có máu **self.health > 0** cũng đồng nghĩa với việc quái vật còn sống

Kiểm tra va chạm tầm nhìn của người chơi so với quái vật:

- Nếu tầm nhìn **self.vision** đang va chạm với kích thước hình ảnh của người chơi **player.rect** và kiểm tra thêm điều kiện người chơi có máu lớn hơn 0 thì ngay lập tức cho trạng thái nghỉ ngoi 'Idle' là True và thời gian nghỉ ngoi là 10 để tránh điều kiện dưới trừ xuống số âm. Đồng thời cho **self.speed = 2** là tốc độ di chuyển của quái vật gấp đôi so với ban đầu cài đặt để quái vật có thể tiến tới người chơi nhanh hơn
- Nếu toạ độ x vị trí ở giữa kích thước hình ảnh của quái vật trừ đi toạ độ x vị trí ở giữa kích thước hình ảnh của người chơi đổi về số dương > 35 thì cập nhật hướng di chuyển cho quái vật đi về phía người chơi
  - if **self.rect.centerx > player.rect.centerx** là vị trí hiện tại ở trục x của người chơi đang đứng sau quái vật thì cập nhật cho chạy về bên trái: **self.moving\_left = True** và **self.moving\_right = False**
  - Ngược lại thì cập nhật **self.moving\_left = False** và **self.moving\_right = True** để quái vật di chuyển về bên phải, hướng người chơi
  - Sau khi kiểm tra xong thì gọi lại phương thức **move()** để quái vật có thể di chuyển về phía người chơi
- Ngược lại với ý 2 ở trên, nếu quái vật đã tiến đến người chơi:
  - Kiểm tra trục y của người chơi và quái vật nếu nhỏ hơn hoặc bằng 30 thì mới được thực hiện hành động tấn công
  - Cập nhật lại biến **self.moving\_left** và **self.moving\_right** cho nó bằng **False** để tránh trường hợp quai vật di chuyển đồng thời để nó đứng im
  - Cập nhật hành động tấn công **self.attack = True**
  - Kiểm tra nếu hành động hiện tại là tấn công thì tiếp đến kiểm tra thêm thời gian sát thương **self.dame\_cooldown == 0** thì cho người chơi nhận 1 lượng sát thương và phải cho biến **self.dame\_cooldown** lại bằng 0 để chờ thời gian gây sát thương tiếp theo
- Nếu chưa thấy người chơi thì cập nhật lại biến **self.speed = 1**

Phần tiếp theo là xét trạng thái nghỉ và cập nhật thời gian nghỉ và số lần di chuyển của quái vật để tạo hiệu ứng đi qua đi lại cho quái vật trở nên chân thật hơn

```

171     if not self.idling and random.randint(1, 200) == 1:
172         self.idling = True
173         self.idling_counter = random.randint(60, 300)
174         self.moving_left = False
175         self.moving_right = False
176     else:
177         if not self.idling:
178             if self.direction == 1:
179                 self.moving_right = True
180                 self.moving_left = False
181             else:
182                 self.moving_right = False
183                 self.moving_left = True
184             self.move(world)
185             self.move_counter += 1
186             if self.move_counter > 32:
187                 self.direction *= -1
188                 self.move_counter *= -1
189             else:
190                 self.idling_counter -= 1
191                 if self.idling_counter <= 0:
192                     self.idling = False

```

### 1. Nếu quái vật chưa nghỉ ngoi và điều kiện random

Nếu quái vật chưa nghỉ ngoi và theo đó là sự kiện random 1 số trong khoảng 1 đến 200 bằng 1 thì sau đó sẽ đổi sang trạng thái nghỉ ngoi **self.idling = True** và cho khoảng nghỉ ngoi từ 1 đến 5 giây và cho các hành động di chuyển **self.moving\_left** và **self.moving\_right = False**

### 2. Nếu quái vật không nghỉ ngoi

- Cập nhật hướng đi:
  - Nếu hướng đi hiện tại **self.direction == 1**, hướng đi bên phải thì cập nhật biến **self.moving\_right = True** và **self.moving\_left = False**
  - Ngược lại nếu hướng đi hiện tại **self.direction == -1**, hướng đi bên trái thì cập nhật lại biến **self.moving\_right = False** và **self.moving\_left = True**

Sau khi cập nhật được 2 biến cho sự di chuyển trái và phải, gọi lại hàm **move()** để giúp quái vật có thể di chuyển. Đồng thời tăng biến đếm số bước đi **self.move\_counter** thêm 1

- Đổi hướng đi:
 

Kiểm tra nếu biến đếm số bước đi lớn hơn 32 thì cập nhật hướng di chuyển lại. Cách đổi là **self.direction \*= -1** và cập nhật lại biến đếm số bước đi **self.move\_counter \*= -1** để quái vật có thể quay lại vị trí ban đầu và tiếp 1 khoảng nữa

### 3. Cập nhật lại thời gian nhàn rỗi

- Biến **self.idling\_counter** được giảm đi 1
- Nếu **self.idling\_counter** nhỏ hơn hoặc bằng 0, tức là đã đếm đủ thời gian bất động, trạng thái bất động quái vật được tắt

```

194     self.rect.x += screen_scroll
195     self.collision_rect.x += screen_scroll

```

Cuối cùng của phương thức **ai()** là cập nhật lại vị trí của hình ảnh cùng với vị trí va chạm 1 khoảng **screen\_scroll** để di lia màn hình thì hình ảnh cũng nhu kích thước của quái vật cũng lia theo

Phương thức **draw()** được sử dụng để vẽ quái vật lên màn hình

```
218     def draw(self, player, screen, screen_scroll, world):
219         self.update()
220         self.update_animation()
221         self.ai(player, screen_scroll, world)
222         screen.blit(pygame.transform.flip(self.image, self.flip, False), self.rect)
223
```

- Gọi phương thức **update()** để cập nhật hành động
- Gọi phương thức **update\_animation()** để cập nhật hoạt ảnh
- **pygame.transform.flip()** được dùng để lật lại hình ảnh của quái vật theo hướng di chuyển
- Hình ảnh của nhân vật **self.image** được vẽ lên màn hình tại vị trí của hình chữ nhật **self.rect**
- Nếu **self.flip** là **True**, tức là quái vật đang di chuyển sang trái, hình ảnh sẽ được lật ngược lại theo chiều ngang

Tiếp đến là file quan trọng nhất trong 3 file. File này dùng để chạy chương trình và em đặt tên cho nó là **main.py**

### 3.4 Cài đặt thư viện, khởi tạo các biến trong cửa sổ trò chơi

#### Khai báo thư viện

```
main.py > ...
1 import pygame
2 import random
3 import csv
4 import button
5 from settings import *
6 import Enemy
```

Câu lệnh *import* để khai báo các thư viện, lần lượt có ý nghĩa như sau

- **pygame** được dùng để tạo game được dùng để tạo game và thao tác đồ họa
- **random** cho phép tạo các số ngẫu nhiên
- **csv** là module để làm việc với các tập tin CSV
- **button** được sử dụng để tạo các nút cho giao diện người dùng trong game
- **Enemy** là lớp quái vật được định nghĩa ở phần 3.3
- **from settings import \*** dùng để nhập tất cả các biến, các hằng số được định nghĩa ở phần 3.1

## Hàm khởi tạo

```
8     pygame.init()
9     pygame.mixer.init()
```

- `pygame.init()` khởi tạo môi trường làm việc cho thư viện Pygame, chuẩn bị cho việc tạo và điều khiển các phần đồ họa và âm thanh cho game
- `pygame.mixer.init()` khởi tạo hệ thống âm thanh của Pygame để sử dụng trong game

## Khởi tạo màn hình chính, màn hình phụ cho cửa sổ trò chơi

```
11    screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
12    surface = pygame.Surface((WIDTH_MAP, HEIGHT_MAP))
13    logo_game = pygame.image.load('Map/Logo/logo_game.png').convert_alpha()
14    pygame.display.set_icon(logo_game)
15    pygame.display.set_caption("Oak Hunter")
```

Khởi tạo màn hình chính của ứng dụng, còn bề mặt **surface** là màn hình phụ dùng ở phần `map_editor()`, đặt tên game là ‘Oak Hunter’ và load icon ở góc bên trái màn hình

## Cài đặt bộ đếm thời gian cho màn hình

```
17    # Bộ đếm thời gian cho màn hình
18    clock = pygame.time.Clock()
```

## Cài đặt phông chữ và các biến để người chơi có thể nâng cấp bản thân

```
20    # Cài đặt phông chữ ThaleahFat
21    font = pygame.font.Font('Map/Font/ThaleahFat.ttf', 18)
22    # Biến dùng để người chơi có thể nâng cấp
23    speed_bullet = 0
24    dame_bullet = 0
25    health_bonus = 0
26    coin_player = 0
27    health_tile = 10
28    bullet_cooldown = 0
```

Loại font chữ được dùng xuyên suốt trong game là loại **TheleahFat.ttf** là cỡ chữ là **18**

Các biến dùng để nâng cấp bản thân trong trò chơi

- **speed\_bullet** là tốc độ đạn của người chơi khi di chuyển
- **dame\_bullet** là lượng dame gây ra của viên đạn áp dụng lên quái vật
- **health\_bonus** là lượng máu tăng thêm mỗi khi update
- **coin\_player** là số lượng đồng xu của người chơi khi thu nhập mỗi map
- **health\_tile** biến này được dùng để chia tỉ lệ máu của người chơi

- bullet cooldown là thời gian để người chơi có thể thực thi phát bắn tiếp theo
- Biến dùng để đánh dấu hành động ở cửa sổ chính game, khi thắng cũng như khi thua trò chơi**

```

29 # Biến dùng để chơi game
30 home_game = True
31 play_game = False
32 option_game = False
33 exit_game = False
34 # Biến dùng khi win game
35 win_game = False

```

- home\_game là biến boolean để kiểm tra có đang ở cửa sổ màn hình trò chơi không
- play\_game là biến boolean kiểm tra xem có đang chơi game không
- option\_game thuộc biến boolean kiểm tra có thực hiện nhiệm vụ trò chơi hay không
- win\_game dùng để kiểm tra người chơi đã thắng game này hay chưa

Tiếp đến là các biến được dùng trong việc tạo map và di chuyển của người chơi lên bản đồ của game

```

36 # Biến dùng trong map editor
37 current_tile = 0
38 scroll = 0
39 scroll_left = False
40 scroll_right = False
41 scroll_speed = 1
42 # Biến dùng để cuộn màn hình
43 screen_scroll = 0
44 bg_scroll = 0
45 level = 1

```

1. Các biến dùng được sử dụng trong hàm **map\_editor()**:

- current\_tile đánh dấu tile hiện tại đang chọn
- scroll để cập nhật ngưỡng cuộn màn hình
- scroll\_left, scroll\_right để kiểm tra đang cuộn sang bên trái hay bên phải
- scroll\_speed là tốc độ của việc cuộn màn hình

2. Các biến được dùng trong việc di chuyển bản đồ của người chơi:

- screen\_scroll là tốc độ cuộn màn hình
- bg\_scroll là tốc độ cuộn của tám nền sau bản đồ
- level là cấp độ bản đồ của người chơi hiện tại

### 3.5 Sử dụng các assets cho hình ảnh, âm thanh và các nút ở trong cửa sổ trò chơi

#### Sử dụng âm thanh trong trò chơi

```
140  """===== Các loại âm thanh trong game ====="""
141  # Tiếng nhận tiền
142  coin_recieved = pygame.mixer.Sound('Audio/coin.wav')
143  coin_recieved.set_volume(0.3)
144  # Tiếng nhảy
145  jump_up = pygame.mixer.Sound('Audio/jump.wav')
146  jump_up.set_volume(0.3)
147  # Tiếng bắn súng
148  shoted = pygame.mixer.Sound('Audio/shot.wav')
149  shoted.set_volume(0.3)
150  # Tiếng đấm
151  punch_audio = pygame.mixer.Sound('Audio/punch.wav')
152  punch_audio.set_volume(0.3)
153  # Tiếng dao
154  slash_audio = pygame.mixer.Sound('Audio/slash.wav')
155  slash_audio.set_volume(0.3)
156  # Tiếng skeleton die
157  skeleton_hurt_audio = pygame.mixer.Sound('Audio/skeletondie.wav')
158  skeleton_hurt_audio.set_volume(0.3)
159  # Tiếng demon die
160  demon_hurt_audio = pygame.mixer.Sound('Audio/demondie.wav')
161  demon_hurt_audio.set_volume(0.3)
162  # Tiếng boss die
163  boss_hurt_audio = pygame.mixer.Sound('Audio/bossdie.wav')
164  boss_hurt_audio.set_volume(0.3)
165  # Tiếng thua game
166  lose_game_audio = pygame.mixer.Sound('Audio/game_over.wav')
167  lose_game_audio.set_volume(0.3)
168  # Tiếng thắng game
169  win_game_audio = pygame.mixer.Sound('Audio/win.wav')
170  win_game_audio.set_volume(0.3)
171  """
172  lose_game_audio_played = False # Dùng để check khi thua
173  win_game_audio_played = False # Dùng để check khi thắng
174  """=====
```

Sử dụng pygame.mixer.Sound để tải các tệp âm thanh từ đường dẫn, cụ thể các file âm thanh này nằm trong folder Audio

Sau đó điều chỉnh âm lượng từng file cho hợp lý với set\_volume của module mixer

**Sử dụng hình ảnh để cài đặt tấm nền bản đồ, thanh máu, thanh đạn và hình ảnh của từng cửa sổ trong trò chơi**

```

47 # Back ground trong game
48 img_1 = pygame.image.load('Map/Backgrounds/1.png').convert_alpha()
49 img_2 = pygame.image.load('Map/Backgrounds/2.png').convert_alpha()
50 img_3 = pygame.image.load('Map/Backgrounds/3.png').convert_alpha()
51 # Biến đổi background sao cho vừa màn hình (1024 x 512)
52 bg_1 = pygame.transform.scale(img_1, (SCREEN_WIDTH, SCREEN_HEIGHT))
53 bg_2 = pygame.transform.scale(img_2, (SCREEN_WIDTH, SCREEN_HEIGHT))
54 bg_3 = pygame.transform.scale(img_3, (SCREEN_WIDTH, SCREEN_HEIGHT))
55 # Bè mặt dùng trong map editor
56 layer_1 = pygame.image.load('Map/Backgrounds/1.png').convert_alpha()
57 layer_2 = pygame.image.load('Map/Backgrounds/2.png').convert_alpha()
58 layer_3 = pygame.image.load('Map/Backgrounds/3.png').convert_alpha()
59 layer_1 = pygame.transform.scale(layer_1, (WIDTH_MAP, HEIGHT_MAP))
60 layer_2 = pygame.transform.scale(layer_2, (WIDTH_MAP, HEIGHT_MAP))
61 layer_3 = pygame.transform.scale(layer_3, (WIDTH_MAP, HEIGHT_MAP))
62
63 empty_health_bar = pygame.image.load('Entity/Player/assets/health_bar.png').convert_alpha()
64 chart_health = pygame.image.load('Entity/Player/assets/chart.png').convert_alpha()
65 board = pygame.image.load('Entity/Player/assets/board.png').convert_alpha()
66 bullet_image = pygame.image.load('Entity/Player/assets/bullet.png').convert_alpha()
67 bullet_image = pygame.transform.scale(bullet_image, (int(bullet_image.get_width() * 0.03), int(bullet_image.get_height() * 0.03)))
68 # board = pygame.transform.scale(board, (int(board.get_width() * 0.8), int(board.get_height() * 0.8)))
69 """===== Các ảnh ở cửa sổ chính ====="""
70 background = pygame.image.load('MainGame/main_game.png').convert_alpha()
71 background = pygame.transform.scale(background, (int(SCREEN_WIDTH), int(SCREEN_HEIGHT)))
72
73 # Cái ảnh settings của game dành cho player
74 setting_game_image = pygame.image.load('MainGame/settings_game.png').convert_alpha()
75 setting_game_image = pygame.transform.scale(setting_game_image, (350, 370))
76 victory_image = pygame.image.load('MainGame/wingame.png').convert_alpha()
77 victory_image = pygame.transform.scale(victory_image, (SCREEN_WIDTH, SCREEN_HEIGHT))
78 """=====

```

Lấy assets từ đường dẫn trong folder **Map** và **MainGame** để tạo hình ảnh cho cửa sổ chính, tâm nền trong trò chơi và trong hàm **map\_editor()** và ảnh thanh máu mà thanh đạn được của người chơi.

## Cài đặt hình ảnh và các nút bấm ở trong cửa sổ chính của trò chơi

```

80 """===== Các nút ở cửa sổ chính ====="""
81 play_btn = pygame.image.load('MainGame/button_action/PlayBtn.png').convert_alpha()
82 play_btn_hover = pygame.image.load('MainGame/button_action/PlayClick.png').convert_alpha()
83 play_btn = pygame.transform.scale(play_btn, (MAINBTN_WIDTH, MAINBTN_HEIGHT))
84 play_btn_hover = pygame.transform.scale(play_btn_hover, (MAINBTN_WIDTH, MAINBTN_HEIGHT))
85
86 exit_btn = pygame.image.load('MainGame/button_action/ExitBtn.png').convert_alpha()
87 exit_btn_hover = pygame.image.load('MainGame/button_action/ExitClick.png').convert_alpha()
88 exit_btn = pygame.transform.scale(exit_btn, (int(MAINBTN_WIDTH), int(MAINBTN_HEIGHT)))
89 exit_btn_hover = pygame.transform.scale(exit_btn_hover, (int(MAINBTN_WIDTH), int(MAINBTN_HEIGHT)))
90
91 option_btn = pygame.image.load('MainGame/button_action/OptBtn.png').convert_alpha()
92 option_btn_hover = pygame.image.load('MainGame/button_action/OptClick.png').convert_alpha()
93 option_btn = pygame.transform.scale(option_btn, (int(MAINBTN_WIDTH), int(MAINBTN_HEIGHT)))
94 option_btn_hover = pygame.transform.scale(option_btn_hover, (int(MAINBTN_WIDTH), int(MAINBTN_HEIGHT)))
95
96 play = button.Button(100, 120, play_btn, play_btn_hover, 1)
97 exit = button.Button(100, 120 + 10 + MAINBTN_HEIGHT, exit_btn, exit_btn_hover, 1)
98 option = button.Button(100, 120 + 10 * 2 + MAINBTN_HEIGHT * 2, option_btn, option_btn_hover, 1)
99 """

```

Sử dụng các hình ảnh từ đường dẫn để hiển thị nút bấm gồm có 3 nút: nút bắt đầu trò chơi, nút chỉnh sửa map và nút thoát trò chơi

Bên cạnh đó dùng thêm **pygame.transform.scale** để điều chỉnh kích thước của các nút

## Cài đặt các nút bên trong trò chơi khi người chơi ấn nút chơi game

```

109  """===== Các nút ở trong game ====="""
110 upgrade_dame = pygame.image.load('MainGame/button_action/dameBullet.png').convert_alpha()
111 upgrade_dame_hover = pygame.image.load('MainGame/button_action/dameHover.png').convert_alpha()
112 upgrade_health = pygame.image.load('MainGame/button_action/healthUpgrade.png').convert_alpha()
113 upgrade_health_hover = pygame.image.load('MainGame/button_action/healthHover.png').convert_alpha()
114 upgrade_speed_bullet = pygame.image.load('MainGame/button_action/speedBullet.png').convert_alpha()
115 upgrade_speed_bullet_hover = pygame.image.load('MainGame/button_action/speedHover.png').convert_alpha()
116 recover_health = pygame.image.load('MainGame/button_action/recoverHealth.png').convert_alpha()
117 recover_health_hover = pygame.image.load('MainGame/button_action/recoverHover.png').convert_alpha()
118 upgrade_cooldown = pygame.image.load('MainGame/button_action/cooldown.png').convert_alpha()
119 upgrade_cooldown_hover = pygame.image.load('MainGame/button_action/cooldownHover.png').convert_alpha()
120 dame_upgrade = button.Button(SCREEN_WIDTH - 50, 200, upgrade_dame, upgrade_dame_hover, 1)
121 health_upgrade = button.Button(SCREEN_WIDTH - 50, 250, upgrade_health, upgrade_health_hover, 1)
122 speed_bullet_upgrade = button.Button(SCREEN_WIDTH - 50, 300, upgrade_speed_bullet, upgrade_speed_bullet_hover, 1)
123 recover_health_upgrade = button.Button(SCREEN_WIDTH - 50, 350, recover_health, recover_health_hover, 1)
124 cooldown_upgrade = button.Button(SCREEN_WIDTH - 50, 400, upgrade_cooldown, upgrade_cooldown_hover, 1)
125
126 home = pygame.image.load('MainGame/button_action/home.png').convert_alpha()
127 home_hover = pygame.image.load('MainGame/button_action/homeHover.png').convert_alpha()
128 home_btn = button.Button(SCREEN_WIDTH - 40, SCREEN_HEIGHT - 40, home, home_hover, 1)
129
130 restartClick = pygame.image.load('MainGame/button_action/restartClick.png').convert_alpha()
131 restartHover = pygame.image.load('MainGame/button_action/restartHover.png').convert_alpha()
132 menuClick = pygame.image.load('MainGame/button_action/menuClick.png').convert_alpha()
133 menuHover = pygame.image.load('MainGame/button_action/menuHover.png').convert_alpha()
134
135 menu_btn = button.Button(SCREEN_WIDTH // 2 - 20 - restartClick.get_width(), SCREEN_HEIGHT // 2 - 100, menuClick, menuHover, 1)
136 restart_btn = button.Button(SCREEN_WIDTH // 2 + 20, SCREEN_HEIGHT // 2 - 100, restartClick, restartHover, 1)
137 """=====

```

Nút **home\_btn** là nút để người chơi khi không muốn chơi game nữa thì có thể thoát game và quay về cửa sổ chính của trò chơi

Các nút **dame\_upgrade**, **health\_upgrade**, **speed\_bullet\_upgrade** và **cooldown\_upgrade** dùng làm các nút bấm khi người chơi nâng cấp bắn thân mình trong trò chơi

Còn 2 nút còn lại: **menu\_btn** và **reset\_btn** là hiển thị khi thắng trò chơi và khi thua trò chơi

### 3.6 Tạo các nhóm sprite có trong trò chơi

```

175 # Nhóm sprite
176 shop_group = pygame.sprite.Group()
177
178 trap_group = pygame.sprite.Group()
179
180 decoration_group = pygame.sprite.Group()
181
182 coin_group = pygame.sprite.Group()
183 level_complete_group = pygame.sprite.Group()
184
185 enemy_group = pygame.sprite.Group()
186

```

- **shop\_group**: nhóm của hàng để người chơi có thể nâng cấp
- **trap\_group**: nhóm bẫy
- **decoration\_group**: nhóm lưu trữ các thành phần trang trí trong game

- **coin\_group**: nhóm đồng xu
- **level\_complete\_group**: nhóm này dùng để đánh dấu người chơi qua được map tiếp theo
- **enemy\_group**: nhóm chứa các kẻ địch mà người chơi phải chiến đấu để vượt qua

Tạo các nhóm dựa trên **pygame.sprite.Group()** để quản lý một tập hợp các **sprite**. **Sprite** là các đối tượng đồ họa có thể được di chuyển, vẽ, hoặc kiểm tra va chạm trong game. Một nhóm cho phép dễ dàng:

- Tổ chức các sprite liên quan
- Cập nhật tất cả sprite trong nhóm cùng lúc bằng lệnh `group.update()`.
- Vẽ tất cả sprite trong nhóm lên màn hình bằng `group.draw(surface)`.
- Kiểm tra va chạm giữa các sprite hoặc giữa các nhóm khác nhau.

### 3.7 Tổ chức tạo các lớp có liên quan tới người chơi

#### Tạo lớp người chơi (Player class)

```

191  """===== Các class trong game ====="""
192  class Player(pygame.sprite.Sprite):
193      def __init__(self, x, y, scale):
194          pygame.sprite.Sprite.__init__(self)
195          animation_types = ['Idle', 'Run', 'Jump', 'Hurt', 'Shot', 'Dead', 'Recharge', 'Walk']
196          self.update_time = pygame.time.get_ticks()
197          self.action = 0
198          self.scale = scale
199          self.frame_index = 0
200          self.animation_list = []
201          for animation in animation_types:
202              sprite_sheet = pygame.image.load(f'Entity/Player/image/{animation}.png').convert_alpha()
203              number_frames = sprite_sheet.get_width() // PLAYER_WIDTH
204              temp_list = []
205              for i in range(number_frames):
206                  frames = sprite_sheet.subsurface(i * PLAYER_WIDTH, 0, PLAYER_WIDTH, PLAYER_HEIGHT)
207                  frames = pygame.transform.scale(frames, (int(PLAYER_WIDTH * self.scale), int(PLAYER_HEIGHT * self.scale)))
208                  temp_list.append(frames)
209              self.animation_list.append(temp_list)

```

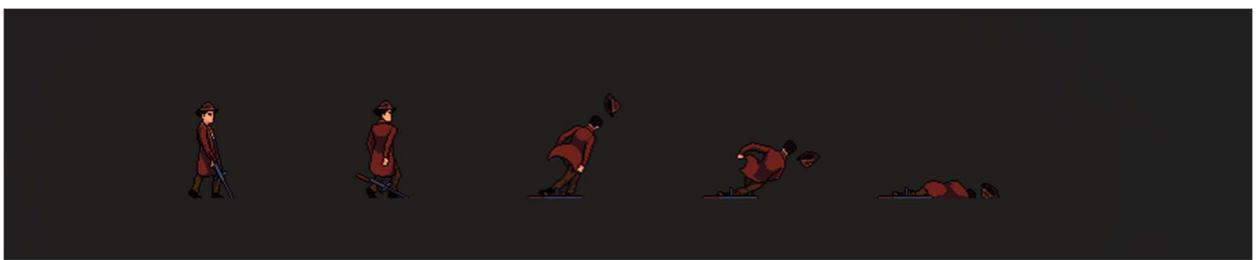
Giống như lớp Enemy, lớp người chơi cũng được định nghĩa vị trí và kích thước của nhân vật. Thay vào đó lớp nhân vật này sẽ có hiệu hành động hơn để phù hợp với tác vụ trong game

Cài đặt hình ảnh cho nhân vật:

1. **animtion\_types** là các loại hành động của nhân vật bao gồm

- **Idle**: nhàn rỗi
- **Run**: chạy
- **Jump**: nhảy lên

- **Shot:** bắn súng
  - **Dead:** chết
  - **Recharge:** thay đạn
  - **Walk:** di chuyển
2. **self.update\_time** là biến dùng để cập nhật thời gian được dùng để thay hoạt ảnh của nhân vật
  3. **self.action** được dùng mục đích đánh dấu hành động của nhân vật
  4. **self.frame\_index** đánh dấu lại đang ở frame thứ mấy trong hành động của nhân vật
  5. **self.animation\_list** là danh sách 2 chiều lưu các hành động của nhân vật



*Minh họa sprite\_sheet của nhân vật gồm có nhiều frames*

Ở vòng lặp for đầu tiên: **for animtion in animation\_types** để duyệt qua từng animtion bên trong **animtion\_types**

Tải hình ảnh **sprite\_sheet** và tính toán số lượng **frame** đang có trong ảnh bằng **number\_frames** với công thức lấy chiều rộng cả ảnh **sprite\_sheet** chia cho chiều rộng của từng **frame**: **sprite\_sheet.get\_widtth() // PLAYER\_WIDTH**

Tiếp tục cắt ảnh và chuyển đổi kích thước hình ảnh bằng **pygame.transform.scale** theo kích thước được định nghĩa sẵn. Cuối cùng là thêm vào danh sách tạm **temp\_list** và thêm danh sách tạm đó vào **self.animation\_list** nhằm lưu trữ các hành động và các **frame** tương ứng với từng loại hành động đó

```

230     # Anh vă vị trí của player
231     self.image = self.animation_list[self.action][self.frame_index]
232     self.rect = self.image.get_rect()
233     self.rect.center = (x, y)
234     # Biến đợi chờ là hành phúc
235     self.shoot_cooldown = 0
236     # Biến kiểm tra va chạm
237     self.width = self.rect.width
238     self.height = self.rect.height
239     self.collision_rect = pygame.Rect(self.rect.centerx - 10 * self.scale, self.rect.bottom - 20 * self.scale, 20 * self.scale, 20 * self.scale)
240     self.coin_collision = pygame.Rect(self.rect.centerx - 10 * self.scale, self.rect.centery, 20 * self.scale, self.rect.height // 2)

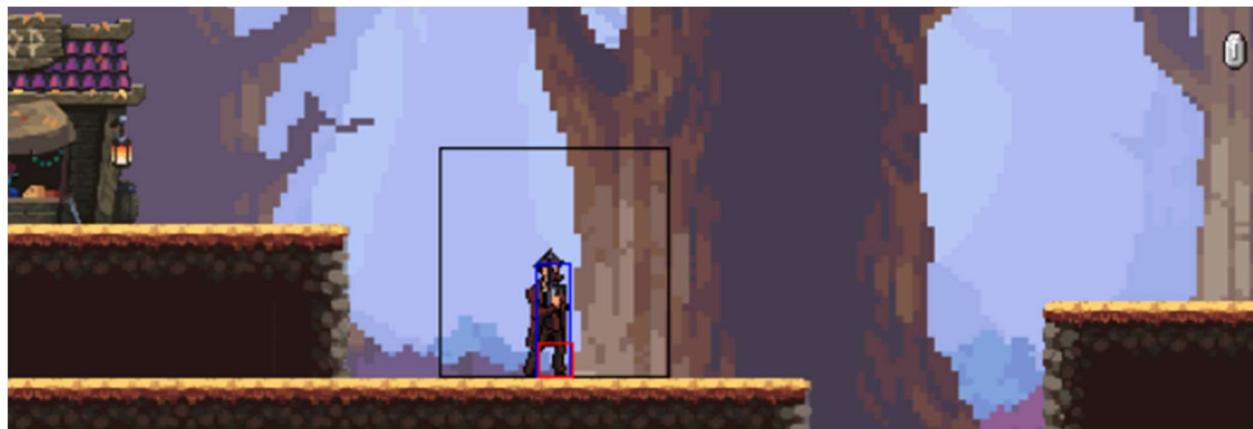
```

Tạo nhóm lưu trữ phần đạn dược của người chơi

```
241 |     # Nhóm đạn được của người chơi  
242 |     self.bullets = pygame.sprite.Group()  
243 |
```

Tạo hình ảnh của nhân vật trên bản đồ cùng với các biến dùng để kiểm tra va chạm của người chơi so với các vật thể có trong game

- **self.image = self.animation\_list[self.action][self.frame\_index]** dùng để khai báo hình ảnh của nhân vật đang nằm ở vị trí nào trong danh sách hoạt ảnh hành động **animation\_list**
- **self.rect** lấy kích thước ảnh của nhân vật và là khối chữ nhật
- **self.rect.center(x, y)** dùng để đặt khối kích thước của nhân vật tại vị trí ở giữa trục x tại x, trục y tại vị trí y
- **self.width** và **self.height** để lấy chiều rộng và chiều cao của nhân vật
- **self.collision\_rect** là biến dùng để kiểm tra sự va chạm của người chơi so với vật cản có trong bản đồ
- **self.coin\_collision** là biến dùng để kiểm tra va chạm của nhân vật với đồng xu có trong game cũng như là kiểm tra va chạm với các loại bẫy và cửa hàng trong trò chơi
- Biến **self.shoot\_cooldown** để cập nhật khoảng thời gian bắn súng của nhân vật



Vẽ lại các khối dùng để kiểm tra va chạm của người chơi

## Các biến người chơi dùng để di chuyển và thực hiện các tác vụ trong game

```
220     # Biến dùng để di chuyển
221     self.moving_left = False
222     self.moving_right = False
223     self.move_jump = False
224     self.run = False
225     self.in_air = False
226     self.hurt = False
227     self.shoot = False
228     self.recharge = False
229     self.vel_y = 0
```

- **self.moving\_left** và **self.moving\_right** là 2 biến boolean để xem người chơi có đi sang bên trái hay đi bên phải hay không
- **self.move\_jump** và **self.in\_air** cũng là 2 biến boolean để cập nhật đang ở dưới đất hay trên trời của người chơi để tránh trường hợp nhân vật có thể nhảy 2 lần mà phải chờ đáp xuống đất mới được nhảy tiếp
- **self.vel\_y** được dùng để áp dụng môi trường trọng lực của nhân vật trong trò chơi
- Các biến boolean được dùng để cập nhật hành động của nhân vật:
  - **self.hurt**: hành động dính sát thương của nhân vật
  - **self.shoot**: hành động bắn súng của nhân vật
  - **self.recharge**: hành động nạp lại đạn của nhân vật
  - **self.run**: hành động chạy của nhân vật

## Cài đặt các biến giới hạn tối đa mà nhân vật có

```
210     # Các biến giới hạn tối đa
211     self.max_health = 150 + health_bonus
212     self.max_bullet = 30
213     self.dame = 20
214     self.speed = 5
215     self.direction = 1
216     self.flip = False
217     # Các biến khởi tạo
218     self.health = self.max_health
219     self.bullet = self.max_bullet
```

- **self.max\_health** ban đầu sẽ là 150 cộng thêm khoảng lượng máu được nâng cấp thông qua các việc nâng cấp của nhân vật
- **self.max\_bullet** định nghĩa lượng đạn được tối đa mà nhân vật có thể mang theo bên mình
- **self.dame** là lượng sát thương ban đầu của viên đạn khi chạm trúng kẻ địch
- **self.speed** là tốc độ di chuyển của người chơi khi di chuyển trên bản đồ
- **self.direction** là hướng đi của nhân vật, mặc định là 1 là bên phải ngược lại -1 là bên trái
- **self.flip** là biến boolean được dùng để lật ảnh của nhân vật lại
- **self.max\_health** là biến khởi tạo lượng máu người chơi hiện tại, mặc định là bằng với lượng máu tối đa **self.max\_health**
- **self.bullet** là lượng đạn hiện tại người chơi đang có, mặc định là bằng với lượng đạn tối đa mà nhân vật có thể mang theo



*Minh họa hình ảnh và t<sup>âm</sup> va chạm của người chơi sau khi cài đặt*

```

243
244     def update_action(self, new_action):
245         if self.action != new_action:
246             self.action = new_action
247             self.frame_index = 0
248

```

Phương thức **update\_action()** này được gọi để khi chuyển hành động này sang hành động khác. Khi chuyển đổi sang hành động mới sẽ reset lại biến ‘self.frame\_index’ lại bằng 0

```

249     def update_animation(self):
250         self.image = self.animation_list[self.action][self.frame_index]
251         # ['Idle', 'Run', 'Jump', 'Hurt', 'Shot', 'Dead', 'Recharge', 'Walk', 'Punch']
252         if pygame.time.get_ticks() - self.update_time > ANIMATION:
253             self.update_time = pygame.time.get_ticks()
254             self.frame_index += 1
255             if self.frame_index >= len(self.animation_list[self.action]):
256                 if self.action == 5: # Dead
257                     self.frame_index = len(self.animation_list[self.action]) - 1
258                 elif self.action == 2: # Jump
259                     self.update_action(0)
260                 elif self.action == 3: # Hurt
261                     self.update_action(0)
262                     self.hurt = False
263                 elif self.action == 4: # Shot
264                     self.update_action(0)
265                     self.shoot = False
266                 elif self.action == 6: # Recharge
267                     self.bullet = self.max_bullet
268                     self.recharge = False
269                     self.update_action(0)
270                 else:
271                     self.frame_index = 0
272

```

Phương thức **update\_animation()** này là phương thức giúp cập nhật hoạt ảnh của nhân vật. Cụ thể

### 1. Cập nhật hình ảnh lại hình ảnh của nhân vật:

Cập nhật hình ảnh bằng cách gọi lại **self.image** sẽ bằng với ảnh trong **self.animation\_list** tại vị trí **self.action** và **self.frame\_index**

### 2. Tăng vị trí frame của ảnh theo thời gian:

Kiểm tra nếu thời gian hiện tại trong game mà trừ với thời gian lúc khởi tạo mà lớn hơn **ANIMATION** thì sẽ cập nhật lại thời gian hiện tại. Đồng thời tăng biến **self.frame\_index** lên thêm 1

### 3. Kiểm tra khi frame của ảnh đang bằng với ảnh cuối cùng trong danh sách chưa hoạt động hiện tại

Ở câu điều kiện thứ 2 **if self.fram\_index >= len(self.animation\_list[self.action])** nghĩa là nếu khung ảnh hiện tại đang lớn hơn cái vị trí của ảnh cuối trong danh sách hành động thì:

- **if self.action == 5:** tức là hành động ‘Dead’ của nhân vật thì sẽ giữ frame cuối lại
- **if self.action == 2:** sau khi chạy hết hoạt động thì cập nhật lại hành động ‘Idle’ tại vị trí 0
- **if self.action == 3:** sau khi chạy hết hoạt động thì cập nhật lại hành động ‘Idle’ và kèm chuyển trạng thái **self.hurt = False**
- **if self.action == 4:** sau khi chạy hết hoạt động thì cập nhật lại hành động ‘Idle’ và kèm chuyển trạng thái **self.shoot = False**
- **if self.action == 6:** sau khi chạy hết hoạt động thì cập nhật lại hành động ‘Idle’ và kèm chuyển trạng thái **self.recharge = False**
- Ngược lại tất cả hành động khác sẽ được thực hiện lặp đi lặp lại sau khi đến vị trí cuối cùng trong danh sách

```
273     def move(self):  
274         screen_scroll = 0  
275         dx = 0  
276         dy = 0  
277  
278         # Điều chỉnh theo hướng di chuyển  
279         if self.run:  
280             dx = (self.speed * self.direction) * 2  
281         if self.moving_right:  
282             dx = self.speed  
283             self.direction = 1  
284             self.flip = False  
285         if self.moving_left:  
286             dx = -self.speed  
287             self.direction = -1  
288             self.flip = True  
289         if self.move_jump and not self.in_air:  
290             self.vel_y = -12  
291             self.move_jump = False  
292             self.in_air = True  
293  
294         # Thêm trọng lực  
295         self.vel_y += GRAVITY  
296         if self.vel_y > 10:  
297             self.vel_y = 10  
298         dy += self.vel_y
```

Sau đó là phương thức **move()** là phương thức di chuyển cho nhân vật

#### 1. Khởi tạo biến:

- **screen\_scroll:** Lưu trữ giá trị cuộn màn hình.

- **dx, dy**: Là các biến di chuyển theo trục x và trục y.
- 2. Xác định hướng di chuyển:**
- Nếu **self.run** là **True** thì **dx** sẽ bằng với tốc độ di chuyển nhân với hướng đi của nhân vật và nhân 2 lén để tốc độ di chuyển nhanh hơn
  - Nếu **self.moving\_left** là **True**, nhân vật di chuyển sang trái, **dx** sẽ được gán là âm, **self.flip** sẽ được đặt thành **True** và **self.direction** được đặt thành -1, ở đây là đi về bên trái.
  - Tương tự, nếu **self.moving\_right** là **True**, nhân vật di chuyển sang phải, **dx** được gán là dương, **self.flip** đặt thành **False** và **self.direction** được đặt thành 1, nhân vật sẽ đi về hướng bên phải.
- 3. Xử lý nhảy:**
- Nếu **self.jump** là **True** và nhân vật không trong trạng thái nhảy (**self.in\_air** là **False**), nhân vật sẽ nhảy lên trên.
  - Thêm vận tốc theo trục y cho nhân vật do tác động của trọng lực. Nếu vận tốc trên trục y lớn hơn 10, nó sẽ được giới hạn lại là 10.
- 4. Áp dụng trọng lực:**
- Vận tốc trục y của nhân vật sẽ được cộng thêm giá trị của hằng số trọng lực **GRAVITY**.
  - Nếu vận tốc trên trục y lớn hơn 10 mức tối đa, nó sẽ không được tăng thêm.
  - Làm như thế để tạo cảm giác trọng lực thực tế, càng rơi xuống lâu vận tốc càng nhanh
- 5. Cập nhật vị trí y của nhân vật:**
- Vị trí y của nhân vật sẽ được cập nhật bằng cách thêm vận tốc trên trục y **self.vel\_y** vào **dy**

```

300     # Kiểm tra va chạm
301     for tile in world.obstacle_list:
302         # Trục x
303         if tile[1].colliderect(self.collision_rect.x + dx, self.collision_rect.y, self.collision_rect.width, self.collision_rect.height):
304             dx = 0
305         # Trục y
306         if tile[1].colliderect(self.collision_rect.x, self.collision_rect.y + dy, self.collision_rect.width, self.collision_rect.height):
307             if self.vel_y < 0:
308                 self.vel_y = 0
309                 dy = tile[1].bottom - self.collision_rect.top
310             elif self.vel_y >= 0:
311                 self.vel_y = 0
312                 self.in_air = False
313                 dy = tile[1].top - self.collision_rect.bottom
314
315         # Ngăn chặn đập đầu dò tường
316         if self.collision_rect.left + dx < 0 or self.collision_rect.right + dx > SCREEN_WIDTH:
317             dx = 0
318         # Kiểm tra coi có bị rớt xuống vực không
319         if self.coin_collision.top > SCREEN_HEIGHT:
320             self.health = 0
321             dy = 0

```

Phần tiếp theo của phương thức **move()** là **xử lý va chạm** giữa nhân vật với các vật khác

1. Kiểm tra va chạm:

- Vòng lặp **for** duyệt qua danh sách **world.obstacle\_list**, kiểm tra va chạm giữa nhân vật và các vật cản.
- Trong mỗi vòng lặp:
  - Kiểm tra va chạm theo hướng x bằng cách sử dụng phương thức **colliderect()**.
  - Nếu có va chạm theo hướng x, **dx** được đặt về 0, ngăn không cho nhân vật tiếp tục di chuyển theo hướng x.
  - Kiểm tra va chạm theo hướng y:
    - Nếu nhân vật đang nhảy lên (**self.vel\_y < 0**), **self.vel\_y** được đặt về 0 và **dy** được điều chỉnh sao cho nhân vật đứng trên mặt đất.
    - Nếu nhân vật đang rơi xuống (**self.vel\_y >= 0**), **self.vel\_y** được đặt về 0, **self.in\_air** được đặt về **False** và **dy** được điều chỉnh sao cho nhân vật đứng dưới mặt đất

## 2. Kiểm tra va chạm với màn hình:

- Kiểm tra đầu tiên **if self.collision\_rect.left + dx < 0 or self.collision\_rect.right + dx > SCREEN\_WIDTH** kiểm tra nếu vị trí va chạm của nhân vật phía bên trái cộng với biến **dx** mà nhỏ hơn 0 hay lớn hơn **SCREEN\_WIDTH** tức là đi về phía rìa bên trái hoặc bên phải màn hình sẽ cập nhật lại **dx** là 0 ngăn cho nhân vật đi ra khỏi khung hình
- Điều kiện thứ 2 kiểm tra xem nhân vật có đi vượt quá phần dưới màn hình không, nếu đi quá thì cập nhật lượng máu của nhân vật về với 0 đồng nghĩa là thua game

```

323     # Cập nhật vị trí của collision_rect
324     self.collision_rect.x += dx
325     self.collision_rect.y += dy
326
327     # Cập nhật vị trí của self.rect để đồng bộ hóa với collision_rect
328     self.rect.centerx = self.collision_rect.centerx
329     self.rect.bottom = self.collision_rect.bottom
330     # Cập nhật vị trí của coin_collision
331     self.coin_collision.x = self.rect.centerx - 10 * self.scale
332     self.coin_collision.y = self.rect.centery
333

```

Tiếp đến là cập nhật lại vị trí của nhân vật cũng như các vị trí xử lí va chạm lại theo biến **dx** và **dy** để nhân vật có thể di chuyển trên màn hình

Cuối cùng phần quan trọng nhất của phương thức **move()** của nhân vật là cập nhật lại biến dùng để cuộn màn hình

```

334     # Cập nhật cuộn màn hình
335     if (self.collision_rect.right > SCREEN_WIDTH - SCROLL_THRESH and bg_scroll < (world.level_length * TILE_SIZE) - SCREEN_WIDTH) or \
336         (self.collision_rect.left < SCROLL_THRESH and bg_scroll > abs(dx)):
337         self.collision_rect.x -= dx
338         screen_scroll = -dx
339
340     return screen_scroll
341

```

- Kiểm tra nếu vị trí va chạm của người chơi hiện tại đang lớn hơn vị trí của ngưỡng màn hình đã cài đặt từ trước nhưng kèm theo đó là biến **bg\_scroll** là biến cuộn **background** phải nhỏ hơn chiều dài của bản đồ **world.level\_length \* TILE\_SIZE** và trừ đi chiều rộng của màn hình **SCREEN\_WIDTH** thì sẽ giữa trừ đi lượng **dx** ban đầu để nhân vật sẽ đứng im nhưng các phần khác vẫn sẽ di chuyển nhờ biến **screen\_scroll** đã được cập nhật theo lượng **-dx**
- Cuối cùng là trả về kết quả cuộn màn hình

```

339     def gun(self):
340         if self.shoot_cooldown == 0 and self.bullet > 0:
341             self.shoot_cooldown = 45 - bullet cooldown
342             bullet = Bullet(self.rect.centerx, self.rect.centery + 20 * self.scale, self.direction, self.flip)
343             self.bullets.add(bullet)
344             self.bullet -= 1
345

```

Sau đó là phương thức **gun()** để nhân vật có thể bắn

- Điều kiện của phương thức này là khi thời gian hồi đạn **shoot\_cooldown** nó phải bằng 0 và số lượng đạn **self.bullet > 0** để ngăn khi không có đạn vẫn bắn được
- Cập nhật lại thời gian nghỉ để hồi đạn là **self.shoot\_cooldown** sẽ bằng với 45 nhưng sẽ trừ đi lượng tốc độ hồi đạn khi nhân vật nâng cấp ở cửa hàng
- Tạo một phần tử từ lớp **Bullet** và thêm nó vào trong nhóm đạn được của người chơi
- Trừ đi lượng đạn đã bắn cho 1

Tiếp sau phương thức **move()** là phương thức dùng để cập nhật hành động cho nhân vật, phương thức **update()**

```

346     def update(self):
347         # ['Idle', 'Run', 'Jump', 'Hurt', 'Shot', 'Dead', 'Recharge', 'Walk', 'Punch']
348         if self.health <= 0:
349             self.health = 0
350             self.update_action[5]
351             self.hurt = False
352             self.moving_left = False
353             self.moving_right = False
354             self.shoot = False
355             self.move_jump = False
356             self.in_air = False
357             self.run = False
358         else:
359             if self.run:
360                 self.update_action(1)
361             elif self.in_air:
362                 self.update_action(2)
363             elif self.hurt:
364                 self.update_action(3)
365             elif self.shoot:
366                 self.update_action(4)
367                 self.gun()
368             elif self.recharge:
369                 self.update_action(6)
370                 # Tất cả hành động khác dừng lại
371                 self.moving_left = False
372                 self.moving_right = False
373                 self.move_jump = False
374                 self.run = False
375                 self.hurt = False
376             elif self.moving_left or self.moving_right:
377                 self.update_action(7)
378             else:
379                 self.update_action(8)
380         if self.shoot_cooldown > 0:
381             self.shoot_cooldown -= 1

```

Tiếp đến là phương thức **update()** có nhiệm vụ là cập nhật lại hành động của kẻ địch cho đúng với trạng thái hiện tại, đồng thời cập nhật lại các biến cooldown

## 1. Cập nhật trạng thái hành động

- Ở câu điều kiện đầu tiên **if self.health <= 0** tức là kiểm tra máu hiện tại nếu nhỏ hơn hoặc bằng 0 thì ngay lập tức cho máu hiện tại bằng 0, **self.health = 0** và cập nhật hành động bằng cách gọi lại phương thức **update\_action()** và truyền cho nó tham số là **5**, tại hành động ‘Death’. Tiếp đến cập nhật toàn bộ biến dùng để kiểm tra đi lại, chạy, nhảy và bắn đạn và nhận sát thương thành **False**
- Ở câu điều kiện ngược lại, **else** kiểm tra hiện nhân vật vẫn còn sống:
  - Nếu đang làm các hoạt động như **self.run**, **self.in\_air**, **self.hurt**, **self.shoot** thì cập nhật cho nó các hành động bằng cách gọi lại hàm **update\_action()** và truyền cho tham số ứng với hành động đó
  - Riêng hành động bắn súng **self.shoot** thì chuyển tất cả hành động khác bằng giá trị **False** để nhân vật có chỉ có thể đứng im nạp đạn
  - Ngược lại nếu không có hành động nào được diễn ra thì cập nhật trạng thái nhàn rỗi ‘Idle’ cho nhân vật

## 2. Cập nhật lại biến cooldown

**if self.shoot\_cooldown > 0** biến **self.shoot\_cooldown** là biến tính thời gian gây sát thương lên người chơi. Nếu biến này lớn hơn 0 thì sẽ được giảm đi 1 mỗi lần gọi hàm **update()**

Vẽ hoạt ảnh và lớp đạn của nhân vật

```
383     def draw(self):
384         self.update_animation()
385         self.update()
386         screen.blit(pygame.transform.flip(self.image, self.flip, False), self.rect)
387         for bullet in self.bullets:
388             bullet.draw()
389             bullet.update()
```

## 3.8 Lớp đạn được (Bullet class)

```

391 class Bullet(pygame.sprite.Sprite):
392     def __init__(self, x, y, direction, flip):
393         pygame.sprite.Sprite.__init__(self)
394         self.speed = 10 + speed_bullet
395         self.image = pygame.image.load('Entity/Player/assets/1.png').convert_alpha()
396         self.image = pygame.transform.scale(self.image, (int(self.image.get_width() * 0.2), int(self.image.get_height() * 0.2)))
397         self.rect = self.image.get_rect()
398         self.rect.center = (x, y)
399         self.direction = direction
400         if flip:
401             self.image = pygame.transform.flip(self.image, True, False)
402
403     def draw(self):
404         self.rect.x += (self.direction * self.speed + speed_bullet) + screen_scroll
405         if self.rect.right < 0 or self.rect.left > SCREEN_WIDTH:
406             self.kill()
407         screen.blit(self.image, self.rect)
408         pygame.draw.rect(screen, 'Black', self.rect, 1)
409

```

Lớp đạn được (**Bullet class**) được dành cho người chơi, nó chính là viên đạn mà người chơi mỗi lần gọi hàm **gun()** mỗi lần gọi lớp này sẽ tương ứng với 1 viên đạn và sẽ được lưu trong biến **bullets** của người chơi

- **\_\_init\_\_(self, x, y, direction, flip)** phương thức khởi tạo của lớp, nhận các tham số sau:
  - **x, y** là toạ độ ban đầu của viên đạn trên màn hình
  - **direction** là hướng đi của viên đạn, hướng đi của người chơi ở đâu thì hướng viên đạn sẽ theo hướng đó
  - **flip** là biến kiểm tra có lật ảnh hay không, ở câu điều kiện trong khởi tạo có gọi biến này nhằm lật lại ảnh của viên đạn cho đúng chiều
  - **self.image** là hình ảnh của viên đạn được tải lên từ thư mục ‘Entity’ và lấy trong thư mục của người chơi phần **assets**. Sau đó được điều chỉnh bằng **pygame.transform.scale** để phù hợp với trò chơi
  - **self.speed** là tốc độ di chuyển của viên đạn, tốc độ này được cộng thêm với một lượng nâng cấp tốc độ viên đạn của người chơi
  - **self.rect** lấy ra được kích thước của viên đạn
- **draw(self)** phương thức này để vẽ và cập nhật lại vị trí của viên đạn  
 $\text{self.rect.x} += (\text{self.direction} * \text{self.speed} + \text{speed_bullet}) + \text{screen\_scroll}$  dùng để cập nhật lại vị trí của hình ảnh viên đạn so với trục x

### 3.9 Lớp Đồng xu (CoinBar class và Coin class)

#### Lớp CoinBar

```

410 class CoinBar(pygame.sprite.Sprite):
411     def __init__(self, x, y):
412         pygame.sprite.Sprite.__init__(self)
413         self.update_time = pygame.time.get_ticks()
414         self.frame_index = 0
415         self.animation_list = []
416         sprite_sheet = pygame.image.load('Entity/Decoration/coin.png').convert_alpha()
417         number_frames = sprite_sheet.get_width() // COIN_WIDTH
418         for i in range(number_frames):
419             frames = sprite_sheet.subsurface(i * COIN_WIDTH, 0, COIN_WIDTH, COIN_HEIGHT)
420             frames = pygame.transform.scale(frames, (int(COIN_WIDTH * 1.5), int(COIN_HEIGHT * 1.5)))
421             self.animation_list.append(frames)
422         self.image = self.animation_list[self.frame_index]
423         self.rect = self.image.get_rect()
424         self.rect.x = x
425         self.rect.y = y
426
427     def draw(self):
428         self.image = self.animation_list[self.frame_index]
429         if pygame.time.get_ticks() - self.update_time > ANIMATION:
430             self.frame_index += 1
431             self.update_time = pygame.time.get_ticks()
432         if self.frame_index >= len(self.animation_list):
433             self.frame_index = 0
434         screen.blit(self.image, self.rect)
435

```

## Lớp Coin

```

489 class Coin(pygame.sprite.Sprite):
490     def __init__(self, x, y):
491         pygame.sprite.Sprite.__init__(self)
492         sprite_sheet = pygame.image.load('Entity/Decoration/coin.png').convert_alpha()
493         self.animation_list = []
494         number_frames = sprite_sheet.get_width() // COIN_WIDTH
495         for i in range(number_frames):
496             frames = sprite_sheet.subsurface(i * COIN_WIDTH, 0, COIN_WIDTH, COIN_HEIGHT)
497             self.animation_list.append(frames)
498         self.frame_index = 0
499         self.image = self.animation_list[self.frame_index]
500         self.rect = self.image.get_rect()
501         self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))
502         self.update_time = pygame.time.get_ticks()
503
504     def draw(self):
505         self.image = self.animation_list[self.frame_index]
506         if pygame.time.get_ticks() - self.update_time > ANIMATION:
507             self.frame_index += 1
508             self.update_time = pygame.time.get_ticks()
509             if self.frame_index >= len(self.animation_list):
510                 self.frame_index = 0
511             self.rect.x += screen_scroll
512         screen.blit(self.image, self.rect)

```

Về phần khởi tạo `__init__` của 2 class dù rằng như không khác nhau là mấy nhưng cả 2 đều có chức năng riêng của mình

- **CoinBar** là class dùng để vẽ thanh đồng xu của nhân vật và cố định không di chuyển
- **Coin** là class dùng cho các đồng xu có mặt trên bản đồ, có thể di chuyển dựa vào biến `screen_scroll`

1. **Khởi tạo:** `__init__(self, x, y)` nhận các tham số sau

- **sprite\_sheet** là sprite sheet chứa 1 hình ảnh nhưng có nhiều frame của ảnh giúp tạo hiệu ứng animation
- **self.animation\_list = []** là danh sách lưu trữ các hoạt ảnh từ sprite\_sheet
- **number\_frames = sprite\_sheet.get\_width() // COIN\_WIDTH** là số lượng frame cắt ra được từ sprite\_sheet
- **self.frame\_index = 0** là vị trí bắt đầu của frame ảnh
- **self.image = self.animation\_list[self.frame\_index]** là hình ảnh của lớp này
- **self.rect = self.image.get\_rect()** lấy ra được kích thước của hình ảnh
- **self.rect.midtop = (x + TILE\_SIZE // 2, y + (TILE\_SIZE - self.image.get\_height()))** đặt vị trí của ảnh ở **midtop** từ biến x và y nhưng có sự điều chỉnh vị trí từ **TILE\_SIZE**
- **self.update\_time = pygame.time.get\_ticks()** biến này dùng để cập nhật thời gian

## 2. Vẽ ảnh và các frame: draw(self)

- **self.image = self.animation[self.frame\_index]** cập nhật lại hình ảnh của lớp này và được lấy ra từ danh sách ở vị trí **self.frame\_index**
- Kiểm tra thời gian hiện tại **pygame.time.get\_ticks()** nếu trù đi thời gian lúc trước **self.update\_time** mà vẫn lớn hơn biến **ANIMATION** thì vị trí hình ảnh sẽ được tăng lên 1 và cập nhật lại thời gian lúc trước thành hiện tại
- Cập nhật lại vị trí x từ **self.rect.x** sẽ cộng với biến **screen\_scroll** để đồng xu có thể di chuyển theo góc di chuyển của nhân vật
- Cuối cùng là vẽ hình ảnh lên bằng **screen.blit(self.image, self.rect)**

## 3.10 Lớp trang trí (Decoration class)

```

466  class Decoration(pygame.sprite.Sprite):
467      def __init__(self, type, img, x, y):
468          pygame.sprite.Sprite.__init__(self)
469          if type != 1:
470              self.image = pygame.image.load(f'Entity/Decoration/{type}.png').convert_alpha()
471          else:
472              self.image = img
473          self.rect = self.image.get_rect()
474          self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))
475
476      def draw(self):
477          self.rect.x += screen_scroll
478          screen.blit(self.image, self.rect)

```

Lớp trang trí có nhiệm vụ vẽ các vật dùng để trang trí lại bản đồ trò chơi, giúp trò chơi trở nên đẹp hơn, cuốn hút hơn

- **\_\_init\_\_(self, type, img, x, y):** nhận các tham số sau

- **x, y** là vị trí của vật phẩm trang trí ở trên bản đồ trò chơi
- Kiểm tra nếu loại != 1 thì là thuộc loại trang trí đặc biệt, nó sẽ lấy ảnh bằng cách pygame.image.load tải ảnh lên từ vị trí khác
- **self.rect** lấy kích thước của vật
- **self.rect.midtop** đặt vị trí ảnh tại điểm x và y nhưng đã được canh chỉnh bằng cách tính toán

- **draw(self)**

- Cập nhật lại vị trí so với trục x **self.rect.x** cộng với một lượng **screen\_scroll**
- Vẽ hình ảnh của vật lên bản đồ **screen.blit(self.image, self.rect)**

### 3.11 Lớp Bẫy (Trap class)

```

514  class Trap(pygame.sprite.Sprite):
515      def __init__(self, type, x, y):
516          # Có 4 loại theo 4 folder
517          pygame.sprite.Sprite.__init__(self)
518          folder = type
519          number_frames = len(os.listdir(f'Entity/Trap/{folder}'))
520          self.animation_list = []
521          self.frame_index = 0
522          for i in range(1, number_frames + 1):
523              frames = pygame.image.load(f'Entity/Trap/{folder}/{i}.png')
524              width = frames.get_width()
525              height = frames.get_height()
526              frames = pygame.transform.scale(frames, (int(width * 0.2), int(height * 0.2)))
527              self.animation_list.append(frames)
528          self.image = self.animation_list[self.frame_index]
529          self.rect = self.image.get_rect()
530          self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))
531          self.update_time = pygame.time.get_ticks()
532          self.dame_cooldown = 0
533
534      def draw(self):
535          self.image = self.animation_list[self.frame_index]
536          if pygame.time.get_ticks() - self.update_time > ANIMATION:
537              self.frame_index += 1
538              self.update_time = pygame.time.get_ticks()
539              if self.frame_index >= len(self.animation_list):
540                  self.frame_index = 0
541          if self.dame_cooldown > 0:
542              self.dame_cooldown -= 1
543          self.rect.x += screen_scroll
544          screen.blit(self.image, self.rect)
545

```

Lớp này sẽ là thử thách dành cho sự khéo léo của người chơi, nếu nhân vật chạm vô sẽ mất máu từ từ, tuy nhiên phần giảm máu của người chơi em sẽ đặt ở một hàm khác và sẽ đê cập sau:

- **\_\_init\_\_(self, type, x, y):** Phương thức khởi tạo của lớp. Nhận các tham số sau:
  - **folder** để xác định loại bẫy gai và lấy đúng sprite\_sheet
  - **sprite\_sheet**: một sprite\_sheet chứa nhiều frame của bẫy gai

- **x, y:** Tọa độ ban đầu của chướng ngại vật trên màn hình.
- **self.image:** hình ảnh của bẫy gai được tải và cắt từ sprite\_sheet theo loại
- **self.rect:** hình chữ nhật giới hạn của chướng ngại vật, được sử dụng để xác định vị trí và va chạm.
- **self.dame\_cooldown** thời gian gây sát thương
- **self.update\_time** lấy thời gian hiện tại để cập nhật được hoạt ảnh tạo thành animation
- **draw(self):** Phương thức cập nhật trạng thái của bẫy gai sau mỗi thời gian và cập nhật lại thời gian gây sát thương lên người chơi. Đồng thời cũng phải cập nhật vị trí trên trục x so với người chơi



*Minh họa 4 loại bẫy gai được sử dụng trong trò chơi*

### 3.11 Lớp Cửa hàng (Shop class)

```

436     class Shop(pygame.sprite.Sprite):
437         def __init__(self, x, y):
438             pygame.sprite.Sprite.__init__(self)
439             sprite_sheet = pygame.image.load('Entity/Decoration/shop.png').convert_alpha()
440             number_frames = sprite_sheet.get_width() // SHOP_WIDTH
441             self.animation_list = []
442             self.frame_index = 0
443             for i in range(number_frames):
444                 frames = sprite_sheet.subsurface(i * SHOP_WIDTH, 0, SHOP_WIDTH, SHOP_HEIGHT)
445                 self.animation_list.append(frames)
446             self.image = self.animation_list[self.frame_index]
447             self.rect = self.image.get_rect()
448             self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))
449             self.update_time = pygame.time.get_ticks()
450
451         def draw(self):
452             self.image = self.animation_list[self.frame_index]
453             if pygame.time.get_ticks() - self.update_time > ANIMATION:
454                 self.frame_index += 1
455                 self.update_time = pygame.time.get_ticks()
456                 if self.frame_index >= len(self.animation_list):
457                     self.frame_index = 0
458             self.rect.x += screen_scroll
459             screen.blit(self.image, self.rect)
460

```

Lớp này là cửa hàng mỗi khi người chơi chạm vào sẽ hiện lên các loại nâng cấp dành cho người chơi, người chơi sẽ chọn các mục tương ứng để nâng cấp bản thân mình trở nên mạnh hơn để chiến đấu với quái vật



*Minh họa hình ảnh cửa hàng*

Việc nâng cấp nhân vật giúp trò chơi trở nên cuốn hút và bớt nhảm chán hơn khi chỉ chiến đấu hoài và không nâng cấp được bản thân mình

Về khởi tạo và vẽ lớp này lên màn hình có phần giống với lớp chông gai nhưng chỉ khác chỗ không có biến đếm ngược thời gian

Về phần va chạm với người chơi để hiển thị thông tin nâng cấp em sẽ làm một hàm riêng để hiển thị thông tin

### 3.12 Lớp cấp độ kế tiếp (NextLevel class)

```
502 class NextLevel(pygame.sprite.Sprite):
503     def __init__(self, x, y, img):
504         pygame.sprite.Sprite.__init__(self)
505         self.image = img
506         self.rect = self.image.get_rect()
507         self.rect.topleft = (x, y)
508
509     def draw(self):
510         self.rect.x += screen_scroll
511         screen.blit(self.image, self.rect)
512
```

Lớp này có nhiệm vụ lưu lại hình ảnh và vị trí nơi mà người chơi chạm phải thì ngay lập tức được dịch chuyển sang cấp độ tiếp theo

Về khởi tạo lớp này có phần khá đơn giản so với các lớp khác

- **\_\_init\_\_(self, x, y, img)** chỉ nhận 3 tham số vị trí và hình ảnh
- **draw(self)** cho phép vẽ hình ảnh lên bản đồ và cập nhật lại vị trí của hình ảnh so với **screen\_scroll**



*Minh họa hình ảnh người chơi đang đứng trước vật dùng để qua màn mới*

Tiếp theo là lớp lớn và quan trọng nhất, lớp này sẽ là lớp phụ trách vị trí, phân loại các ô và vẽ ra bản đồ cho từng loại level của game

### 3.13 Lớp thế giới (World class)

```
... ----- Dữ liệu thế giới ----- ...
class World:
    def __init__(self):
        self.obstacle_list = []
```

#### Khởi tạo

**\_\_init\_\_(self)** lớp này chỉ lưu lại 1 dữ liệu đó là **self.obstacle\_list** nó chính là danh sách lưu lại các bờ mặt bản đồ, nơi mà người chơi sẽ tiếp xúc và di chuyển trên đó

#### Đọc và lấy dữ liệu

```

def process_data(self, world_data):
    self.level_length = len(world_data[0])
    player = None
    for y, row in enumerate(world_data):
        for x, tile in enumerate(row):
            if tile >= 0:
                img = img_list[tile]
                img_rect = img.get_rect()
                img_rect.x = x * TILE_SIZE
                img_rect.y = y * TILE_SIZE
                tile_data = (img, img_rect) # Tuple
                if (tile >= 0 and tile <= 4) or (tile >= 9 and tile <= 17) or (tile >= 19 and tile <= 20):
                    self.obstacle_list.append(tile_data)
            elif tile == 22:
                rock = Decoration('rock_1', img, img_rect.x, img_rect.y)
                decoration_group.add(rock)
            elif tile == 23:
                rock = Decoration('rock_2', img, img_rect.x, img_rect.y)
                decoration_group.add(rock)
            elif tile == 24:
                rock = Decoration('rock_3', img, img_rect.x, img_rect.y)
                decoration_group.add(rock)
            # Grass
            elif tile == 25:
                grass = Decoration('grass_1', img, img_rect.x, img_rect.y)
                decoration_group.add(grass)
            elif tile == 26:
                grass = Decoration('grass_2', img, img_rect.x, img_rect.y)
                decoration_group.add(grass)
            elif tile == 27:
                grass = Decoration('grass_3', img, img_rect.x, img_rect.y)
                decoration_group.add(grass)
            # Lamp
            elif tile == 28:
                lamp = Decoration('lamp', img, img_rect.x, img_rect.y)
                decoration_group.add(lamp)
            # Fence
            elif tile == 29:
                fence = Decoration('fence_1', img, img_rect.x, img_rect.y)
                decoration_group.add(fence)
            elif tile == 30:
                fence = Decoration('fence_2', img, img_rect.x, img_rect.y)

```

### process\_data(self, data):

- Phương thức này được sử dụng để xử lý dữ liệu đầu vào và tạo ra thế giới dựa trên dữ liệu đó.
- Tham số **data** là dữ liệu đầu vào để tạo thế giới.
- Dòng **self.level\_length = len(data[0])** tính độ dài của mỗi hàng trong dữ liệu, giả sử tất cả các hàng có cùng độ dài.
- Duyệt qua từng hàng và từng ô của dữ liệu:
  - Nếu giá trị của ô không âm (**tile >= 0**), thực hiện các hành động sau:
    - Tạo một hình ảnh và một hình chữ nhật hình ảnh (**img\_rect**) dựa trên giá trị của ô đó trong **img\_list**.
    - Đặt vị trí của hình chữ nhật hình ảnh tại vị trí tương ứng trên màn hình.
    - Tạo một tuple chứa hình ảnh và hình chữ nhật hình ảnh và thêm nó vào **self.obstacle\_list**.
  - Nếu giá trị của ô nằm trong phạm vi từ 0 đến 4 hoặc 9 đến 17 và 19 hoặc 20, nó sẽ lưu lại đây sẽ là bề mặt tiếp xúc của người chơi với các bề mặt trong trò chơi

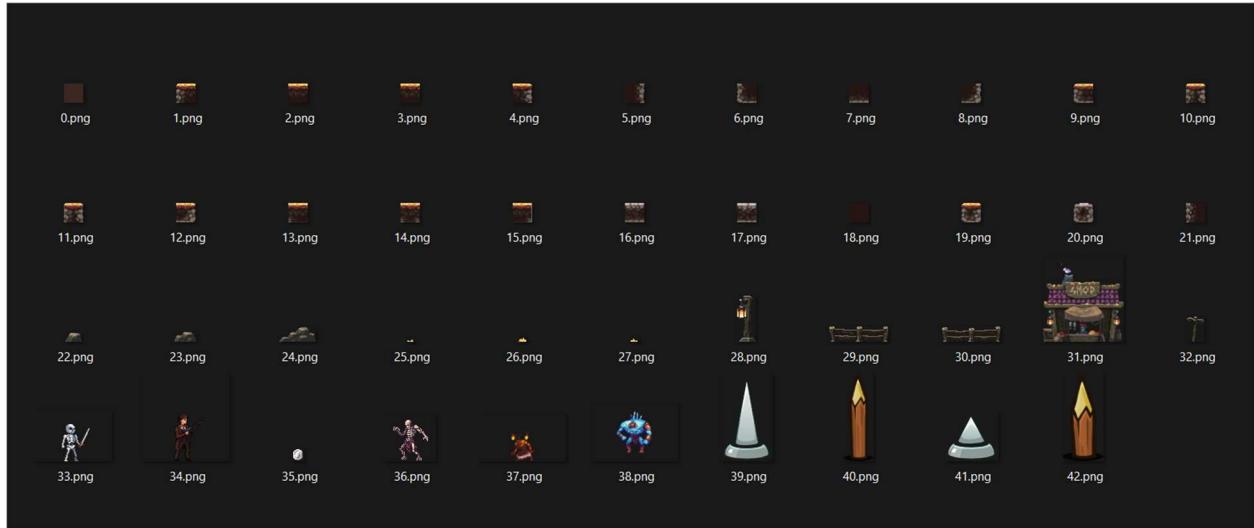
- Nếu giá trị của ô là từ 22 đến 30 nó sẽ tạo ra các lớp trang trí đặc biệt thuộc đối tượng ‘Decoration’ và lưu vào **decoration\_group**
- Nếu ô có giá trị là 30 sẽ tạo một đối tượng ‘Shop’ và lưu trữ trong **shop\_group**

```

594         elif tile == 31: # Shop
595             shop = Shop(img_rect.x, img_rect.y)
596             shop_group.add(shop)
597             # Next Level
598         elif tile == 32:
599             next_level = NextLevel(img_rect.x, img_rect.y, img)
600             level_complete_group.add(next_level)
601         elif tile == 33: # Skeleton
602             enemy = Enemy.Enemy('Skeleton', img_rect.x, img_rect.y, 1)
603             enemy_group.add(enemy)
604         elif tile == 34: # Player
605             player = Player(img_rect.x, img_rect.y, 0.75)
606             # Coin
607         elif tile == 35:
608             coin = Coin(img_rect.x, img_rect.y)
609             coin_group.add(coin)
610         elif tile == 36:
611             enemy = Enemy.Enemy('Bigger', img_rect.x, img_rect.y, 1)
612             enemy_group.add(enemy)
613         elif tile == 37:
614             enemy = Enemy.Enemy('Demon', img_rect.x, img_rect.y, 1)
615             enemy_group.add(enemy)
616         elif tile == 38:
617             enemy = Enemy.Enemy('Boss', img_rect.x, img_rect.y, 1)
618             enemy_group.add(enemy)
619         elif tile == 39:
620             traps = Trap('long_metal', img_rect.x, img_rect.y)
621             trap_group.add(traps)
622         elif tile == 40:
623             traps = Trap('long_wood', img_rect.x, img_rect.y)
624             trap_group.add(traps)
625         elif tile == 41:
626             traps = Trap('small_metal', img_rect.x, img_rect.y)
627             trap_group.add(traps)
628         elif tile == 42:
629             traps = Trap('small_wood', img_rect.x, img_rect.y)
630             trap_group.add(traps)
631         else:
632             decoration = Decoration(1, img, img_rect.x, img_rect.y)
633             decoration_group.add(decoration)

```

- Các ô có giá trị từ 39 đến 42 là các ô thuộc lớp bẫy nó sẽ tạo ra đối tượng ‘Trap’ và lưu trong **trap\_group**
- Các ô có giá trị 33, 36, 37, 38 là các ô quái vật, mỗi ô là một loại quái vật, các ô này đều thuộc đối tượng ‘Enemy’ và được lưu trong cùng 1 nhóm là **enemy\_group**
- Ô thứ 35 là ô đồng xu, khi xét điều kiện đúng với ô này nó sẽ tạo ra một đối tượng đồng xu và lưu trong **coin\_group**
- Ô thứ 34 là ô của người chơi, nó sẽ tạo ra lớp người chơi ứng với vị trí đó và trả về lại sau khi gọi hàm **process\_data()**
- **return player**
- Ngược lại các giá trị còn lại sẽ thuộc lớp trang trí bình thường và được lưu trong **decoration\_group**



*Hình ảnh các tile*

```
def draw(self):
    for tile in self.obstacle_list:
        tile[1].x += screen_scroll
        screen.blit(tile[0], tile[1])
```

Phương thức **draw(self)** của lớp **World** được sử dụng để vẽ các bê mặt trong game lên màn hình.

- Dùng vòng lặp **for** để duyệt qua mỗi vật cản trong danh sách **obstacle\_list**
- Mỗi vật cản được biểu diễn bởi một bộ dữ liệu **tile**, gồm hình ảnh của vật cản (**tile[0]**) và hình chữ nhật giới hạn của nó (**tile[1]**)
- Vị trí của vật cản được cập nhật bằng cách thêm giá trị của **screen\_scroll** vào tọa độ x của hình chữ nhật giới hạn của nó (**tile[1][0] += screen\_scroll**)
- Cuối cùng, vật cản được vẽ lên màn hình bằng cách sử dụng hàm **blit** của Pygame

### 3.14 Khởi tạo các mảng lưu trữ các thành phần của thế giới

```

642 """
643     --- Chuẩn bị thành phần cho main game ---
644
645     # Tao mang luu cac tile trong map
646     img_list = []
647     for i in range(TILE_TYPES):
648         img = pygame.image.load(f'Map/Tile/{i}.png').convert_alpha()
649         img = pygame.transform.scale(img, (TILE_SIZE, TILE_SIZE))
650         img_list.append(img)
651
652     # Luu tru cac nut trong map editor
653     button_list = []
654     button_col = 0
655     button_row = 0
656     for i in range(len(img_list)):
657         tile_button = button.Button(WIDTH_MAP + 10 + button_col * TILE_SIZE + button_col * 10, 10 + button_row * TILE_SIZE + button_row * 10, img_list[i], img_list[i], 1)
658         button_list.append(tile_button)
659         button_col += 1
660         if button_col == 7:
661             button_row += 1
662             button_col = 0
663
664     # Tao mang data rong dung de chua du lieu the gioi
665     world_data = []
666     for row in range(ROWS):
667         r = [-1] * COLS
668         world_data.append(r)
669     # Doc du lieu len mang vua tao
670     with open(f'Level/level{level}_data.csv', newline = '') as csvfile:
671         reader = csv.reader(csvfile, delimiter = ',')
672         for x, row in enumerate(reader):
673             for y, tile in enumerate(row):
674                 world_data[x][y] = int(tile)

```

Đầu tiên là lưu các tile vào một danh sách đặt tên là **img\_list**, danh sách này sẽ lưu trữ các ô hình ảnh đồ họa có trong folder, tải lên bằng **image.load** của **pygame** và sẽ được dùng để lấy ảnh ra và vẽ lên màn hình

Tiếp đến là lưu trữ các nút bấm của từng loại tile, 1 tile sẽ có cho mình loại nút bấm riêng, để phân loại thì mỗi tile sẽ lưu trữ 1 loại hình ảnh khác nhau, em đặt tên cho nó là **button\_list**

Cuối cùng là tạo 1 list **world\_data** rỗng sau đó lấy thông tin chiều cao và chiều dài của bản đồ màn chơi hiện tại

Giờ sẽ tải thế giới từ file csv trong thư mục Level. Vòng lặp điều kiện dùng để đọc dữ liệu từng ô

A1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
8	-1	-1	34	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	35	-1	-1	
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	33	-1	
10	-1	28	-1	-1	31	-1	-1	-1	42	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
11	1	2	2	2	2	3	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
12	21	18	18	18	18	18	5	-1	42	-1	39	-1	41	-1	-1	-1	1	2	2	2	2	
13	21	18	18	18	1	2	3	2	2	2	2	2	4	-1	-1	-1	21	18	18	18	18	
14	21	18	18	21	18	18	18	18	18	18	18	18	5	-1	-1	-1	21	18	18	18	18	
15	21	18	18	21	18	18	18	18	18	18	18	18	5	-1	-1	-1	21	18	18	18	18	
16	6	7	7	21	18	18	18	18	18	18	18	18	5	-1	-1	-1	21	18	1	2	2	

Ví dụ về cấu trúc file csv:

- 1 là ô rỗng, các ô > -1 tương đương với ô được tạo trong class World
- Sau đó trả về thônh tin người chơi

### 3.15 Các hàm hỗ trợ trong trò chơi

Hàm vẽ chữ **draw\_text()**

```
677  # Hàm vẽ chữ
678  def draw_text(text, font, color, x, y):
679      # Transform string to img
680      img = font.render(text, True, color)
681      screen.blit(img, (x, y))
682
```

Hàm nhận các tham số:

- **text** là chữ, nó sẽ được chuyển từ dạng string sang img bằng cách gọi **img = font.render(text, True, color)**
- **font** là phông chữ
- **color** là màu sắc của chữ
- **x, y** là 2 biến tọa độ x và y

Hàm vẽ tâm nền cho trò chơi **draw\_bg()**

```
683  # Hàm vẽ background để tạo hiệu ứng 3D dựa trên các layer
684  def draw_bg():
685      width = bg_1.get_width()
686      for i in range(20):
687          screen.blit(bg_1, ((i * width) - bg_scroll * 0.5, 0))
688          screen.blit(bg_2, ((i * width) - bg_scroll * 0.6, 0))
689          screen.blit(bg_3, ((i * width) - bg_scroll * 0.7, 0))
690
```

Hàm này được dùng để vẽ tâm nền cho toàn bộ bản đồ của mỗi vòng chơi, gồm có 3 layer được đặt là **bg\_1**, **bg\_2**, **bg\_3** và sẽ được nhân theo tỉ lệ của **bg\_scroll** nhằm tạo ra hiệu ứng 3D cho tâm nền. Điều này giúp tâm nền game trở nên chân thật và sinh động hơn

Hàm reset dữ liệu **reset\_data()**

```
700  def reset_data():
701      global coin_player
702      decoration_group.empty()
703      shop_group.empty()
704      coin_group.empty()
705      enemy_group.empty()
706      level_complete_group.empty()
707      trap_group.empty()
708      coin_player = 0
709
```

Hàm này sinh ra để người dùng có thể tái gọi phương thức, nó sẽ reset toàn bộ dữ liệu có trong các **sprite.group.Group** và reset lại dữ liệu đồng xu của người chơi khi chiến thắng game hay qua màn

Kế tiếp là các hàm dùng để vẽ quái vật, các nhóm và biểu đồ của người chơi

```

710  def draw_enemy(player):
711    for enemy in enemy_group:
712      enemy.draw(player, screen, screen_scroll, world)
713      if enemy.attack == True and enemy.dame_cooldown == 90:
714        if enemy.type == 'Skeleton' or enemy.type == 'Demon':
715          slash_audio.play()
716        else:
717          punch_audio.play()
718
719  def draw_group(player):
720    for decoration in decoration_group:
721      decoration.draw()
722    for next_level in level_complete_group:
723      next_level.draw()
724    for shop in shop_group:
725      shop.draw()
726    for coin in coin_group:
727      coin.draw()
728    for trap in trap_group:
729      trap.draw()
730      if trap.rect.colliderect(player.coin_collision):
731        if trap.dame_cooldown == 0:
732          player.health -= 10
733          trap.dame_cooldown = 20
734
735  def draw_chart(player):
736    screen.blit(empty_health_bar, (10, 10))
737    draw_text(f'{coin_player}', font, WHITE, 15 + CoinBarPlayer.image.get_width(), 79)
738    number_tiles = player.health // health_tile
739    step = 0
740    for i in range(number_tiles):
741      if i == 5:
742        step = 10
743      if i == 10:
744        step = 21
745      screen.blit(chart_health, (76 + i * 16 + step, 33))
746    for i in range(player.bullet):
747      screen.blit(bullet_image, (10 + i * 10, 100))

```

1. Hàm vẽ quái vật **draw\_enemy()** hàm này sẽ duyệt qua từng loại quái vật có trong **enemy\_group** sau đó vẽ các loại quái vật tương ứng lên màn hình. Nếu quái vật đang tấn công và thời gian gây sát thương **dame\_cooldown** đang là 50 thì kích hoạt 2 loại âm thanh khác nhau tương ứng vs quái vật khác nhau
2. Hàm vẽ các vật phẩm có trong **sprite.group.Group** tên là **draw\_group()**. Ở mỗi vòng lặp for sẽ duyệt qua từng đối tượng có trong đó và sẽ vẽ lên bàn đạp. Riêng các đối tượng ‘Trap’ sẽ gây sát thương cho người chơi mỗi lần **dame\_cooldown** và biến này sẽ được cập nhật lại sau mỗi lần gây sát thương
3. Hàm vẽ thanh máu và thanh đạn của người chơi có tên là **draw\_chart()** được dùng để vẽ thanh máu cùng với thanh đạn, tỉ lệ thanh máu được chia phù hợp với biến **health\_tile** để tránh máu tràn viền

Tiếp tục là các hàm dùng để kiểm tra va chạm

```

751 def bullet_enemy(player):
752     for bullet in player.bullets:
753         for enemy in enemy_group:
754             if bullet.rect.colliderect(enemy.collision_rect):
755                 if enemy.health > 0:
756                     enemy.health -= player.dame + dame_bullet
757                 if enemy.health <= 0:
758                     if enemy.type == 'Skeleton' or enemy.type == 'Bigger':
759                         skeleton_hurt_audio.play()
760                     elif enemy.type == 'Demon':
761                         demon_hurt_audio.play()
762                     else:
763                         boss_hurt_audio.play()
764                     if random.randint(1, 3) == 1:
765                         enemy.hurt = True
766                     player.bullets.remove(bullet)
767
768 def check_enemy_alive():
769     number_of_enemy = 0
770     for enemy in enemy_group:
771         if enemy.health == 0:
772             number_of_enemy += 1
773     return number_of_enemy == len(enemy_group)
774
775 def coin_collision(player):
776     global coin_player
777     for coin in coin_group:
778         if coin.rect.colliderect(player.coin_collision):
779             coin_group.remove(coin)
780             coin_player += 1
781             coin_recieved.play()
782

```

1. Hàm đầu tiên là hàm **bullet\_enemy()** hàm này sẽ duyệt qua từng đối tượng ‘Bullet’ có trong lớp người chơi, kiểm tra nếu ảnh viên đạn va chạm với quái vật mà còn sống thì nó sẽ tự động mất đi. Đồng thời trừ một lượng máu tương ứng. Nếu quái vật có lượng máu nhỏ hơn hoặc bằng 0 thì đồng nghĩa quái vật ‘Die’ và cập nhật âm thanh tương ứng. Nếu **random** được đúng số thì cập nhật hành động nhận sát thương. Việc **random** này nhằm để tranh việc quái vật cứ dừng lại và nhận sát thương
2. Hàm kế tiếp là **check\_enemy\_alive()** mục đích của hàm này là để kiểm tra số quái vật trong bản đồ đang có số máu bằng 0 thì mới cho qua màn
3. Hàm kế tiếp là **coin\_collision()** hàm này để kiểm tra sự va chạm đồng xu với người chơi. Nếu người chơi chạm đồng xu thì nó sẽ chạy âm thanh và số đồng xu người chơi được cộng thêm 1

```

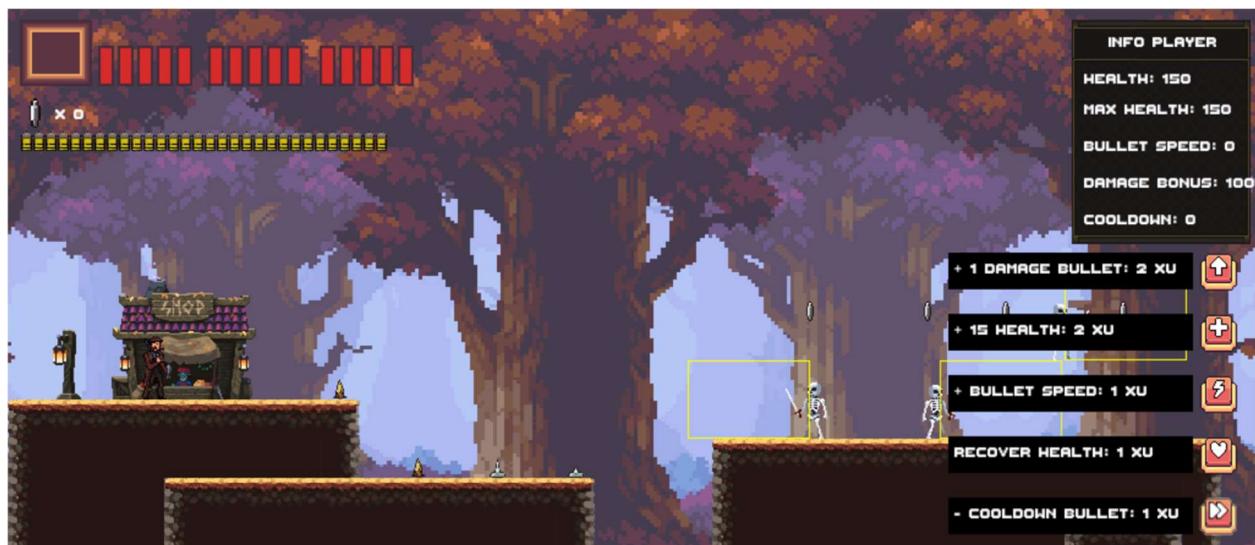
783     def shop_collision(player):
784         for shop in shop_group:
785             if shop.rect.colliderect(player.coin_collision):
786                 # Hiển thị bảng cửa hàng
787                 screen.blit(board, (870, 10))
788                 draw_text('Info Player', font, 'White', 900, 20)
789                 draw_text(f'Health: {player.health}', font, 'White', 880, 50)
790                 draw_text(f'Max Health: {player.max_health}', font, 'White', 880, 75)
791                 draw_text(f'Bullet Speed: {speed_bullet}', font, 'White', 880, 105)
792                 draw_text(f'Damage Bonus: {dame_bullet}', font, 'White', 880, 135)
793                 draw_text(f'Cooldown: {bullet_cooldown}', font, 'White', 880, 165)
794
795                 # Thêm thông tin trên mỗi nút
796                 pygame.draw.rect(screen, "Black", (SCREEN_WIDTH - 255, 200, 200, 30))
797                 draw_text("+ 1 Damage Bullet: 2 xu", font, "White", SCREEN_WIDTH - 250, 205)
798             if dame_upgrade.draw(screen):
799                 handle_upgrade("dame", player)
800                 pygame.draw.rect(screen, "Black", (SCREEN_WIDTH - 255, 250, 200, 30))
801                 draw_text("+ 15 Health: 2 xu", font, "White", SCREEN_WIDTH - 250, 255)
802             if health_upgrade.draw(screen):
803                 handle_upgrade("health", player)
804                 pygame.draw.rect(screen, "Black", (SCREEN_WIDTH - 255, 300, 200, 30))
805                 draw_text("+ Bullet Speed: 1 xu", font, "White", SCREEN_WIDTH - 250, 305)
806             if speed_bullet_upgrade.draw(screen):
807                 handle_upgrade("speed_bullet", player)
808                 pygame.draw.rect(screen, "Black", (SCREEN_WIDTH - 255, 350, 200, 30))
809                 draw_text("Recover Health: 1 xu", font, "White", SCREEN_WIDTH - 250, 355)
810             if recover_health_upgrade.draw(screen):
811                 handle_upgrade("recover_health", player)
812                 pygame.draw.rect(screen, "Black", (SCREEN_WIDTH - 255, 400, 200, 30))
813                 draw_text("- Cooldown Bullet: 1 xu", font, "White", SCREEN_WIDTH - 250, 405)
814             if cooldown_upgrade.draw(screen):
815                 handle_upgrade(["cooldown", player])
816
817     def handle_upgrade(upgrade_type, player):
818         """
819             Xử lý logic nâng cấp dựa trên loại nâng cấp.
820         """
821         global coin_player, dame_bullet, speed_bullet, bullet_cooldown, max_health_current, health_tile
822         if upgrade_type == "dame" and coin_player >= 2:
823             dame_bullet += 1
824             coin_player -= 2
825         elif upgrade_type == "health" and coin_player >= 2:
826             player.max_health += 15
827             health_tile += 1
828             coin_player -= 2
829         elif upgrade_type == "speed_bullet" and coin_player >= 1 and speed_bullet < 5:
830             speed_bullet += 1
831             coin_player -= 1
832         elif upgrade_type == "recover_health" and coin_player >= 1:
833             player.health = player.max_health
834             coin_player -= 1
835         elif upgrade_type == "cooldown" and coin_player >= 1 and bullet_cooldown < 35:
836             bullet_cooldown += 1
837             coin_player -= 1

```

Đây là hàm kiểm tra va chạm của người chơi với các cửa hàng và em đặt tên là **shop\_collision()**

- Mỗi lần chạm cửa hàng sẽ hiện ra thông tin người chơi cùng với đó là các nâng cấp mà người chơi có thể chọn

- Với số lượng xu tương ứng người chơi có thể nâng cấp các thuộc tính của mình để trở nên mạnh hơn
- Giới hạn số lượt nâng cấp để ngăn ngừa việc người chơi quá mạnh dẫn đến trò chơi trở nên chán đi
- Các loại nâng cấp của người chơi được đề cập
  - Nâng cấp sát thương của đạn: giá 2 xu
  - Nâng cấp thêm máu tối đa cho người chơi: 2 xu
  - Nâng cấp tốc độ đạn di chuyển: 1 xu
  - Hồi lập tức 100% máu: 1 xu
  - Giảm thời gian hồi lượt bắn đạn: 1 xu



*Minh họa khi người chơi va chạm với cửa hàng*

### 3.16 Hàm xây dựng cửa sổ trò chơi

Trong mỗi tựa game chơi, đều phải có phần cài đặt bản đồ thì mới có thể chơi được. Chính vì thế em đã thiết kế thêm 1 phương thức giúp người chơi có thể điều chỉnh bản đồ chơi theo nhu mong muốn và em đặt tên cho nó là **map\_editor()**

```

846 def map_editor():
847     action_edit = True
848     global current_tile
849     global scroll
850     global scroll_left
851     global scroll_right
852     global scroll_speed
853     global home_game
854     global level
855     global MAX_LEVELS
856     global player
857     screen.fill(GRAY)

```

## Khởi tạo các biến toàn cục:

- Các biến như `current_tile`, `scroll`, `scroll_left`, `scroll_right`, `scroll_speed`, `home_game`, `level` và `MAX_LEVEL` được sử dụng để khai báo rằng biến đó là một biến toàn cục (global variable) và có thể được truy cập hoặc sửa đổi từ bất kỳ đâu trong chương trình, miễn là biến đó đã được khai báo trước đó trong phạm vi toàn cục (global scope)
- Hàm `screen.fill(GRAY)` được sử dụng để xoá màn hình với màu xám

Sau đó là vẽ background cho phần chỉnh sửa bản đồ

```
864     # Vẽ cái nền chỗ cso thêm tile vào map
865     num_tiles = (COLS * TILE_SIZE) // WIDTH_MAP + 1
866     for i in range(num_tiles):
867         surface.blit(layer_1, ((i * WIDTH_MAP) - scroll * 0.5, 0))
868         surface.blit(layer_2, ((i * WIDTH_MAP) - scroll * 0.6, 0))
869         surface.blit(layer_3, ((i * WIDTH_MAP) - scroll * 0.7, 0))
```

- Đầu tiên tính số lượng lặp để vẽ tám nền theo công thức `num_tiles = (COLS * TILE_SIZE) // WIDTH_MAP + 1` để đảm bảo sẽ vẽ hết bản đồ trò chơi
- Sau đó các tám nền sẽ được vẽ lên bề mặt tạm thời `surface` theo từng layer, ngoài ra để tạo thêm hiệu ứng 3D em đã điều chỉnh tỉ lệ của scroll để cho phép tám nền trở nên sinh động hơn

Tiếp đến là vẽ lối đi để điều chỉnh ô

```
870     # Vẽ cái lưới chỗ map
871     for i in range(ROWS + 1):
872         pygame.draw.line(surface, (255, 255, 255), (0, i * TILE_SIZE_MAP ), (WIDTH_MAP, i * TILE_SIZE_MAP ))
873     for j in range(COLS + 1):
874         pygame.draw.line(surface, (255, 255, 255), (j * TILE_SIZE_MAP - scroll, 0), (j * TILE_SIZE_MAP - scroll, HEIGHT_MAP ))
```

- Với các đường ngang, vòng lặp chạy từ `0` đến `ROWS` và vẽ mỗi đường ngang tại vị trí `(0, i * TILE_SIZE_MAP)` đến `(WIDTH_MAP, i * TILE_SIZE_MAP)`
- Với các đường dọc, vòng lặp chạy từ `0` đến `COLS` và vẽ mỗi đường dọc tại vị trí `(j * TILE_SIZE_MAP - scroll, 0)` đến `(j * TILE_SIZE_MAP - scroll, HEIGHT_MAP)`
- Biến `scroll` được sử dụng để di chuyển lối đi theo vị trí cuộn của màn hình.

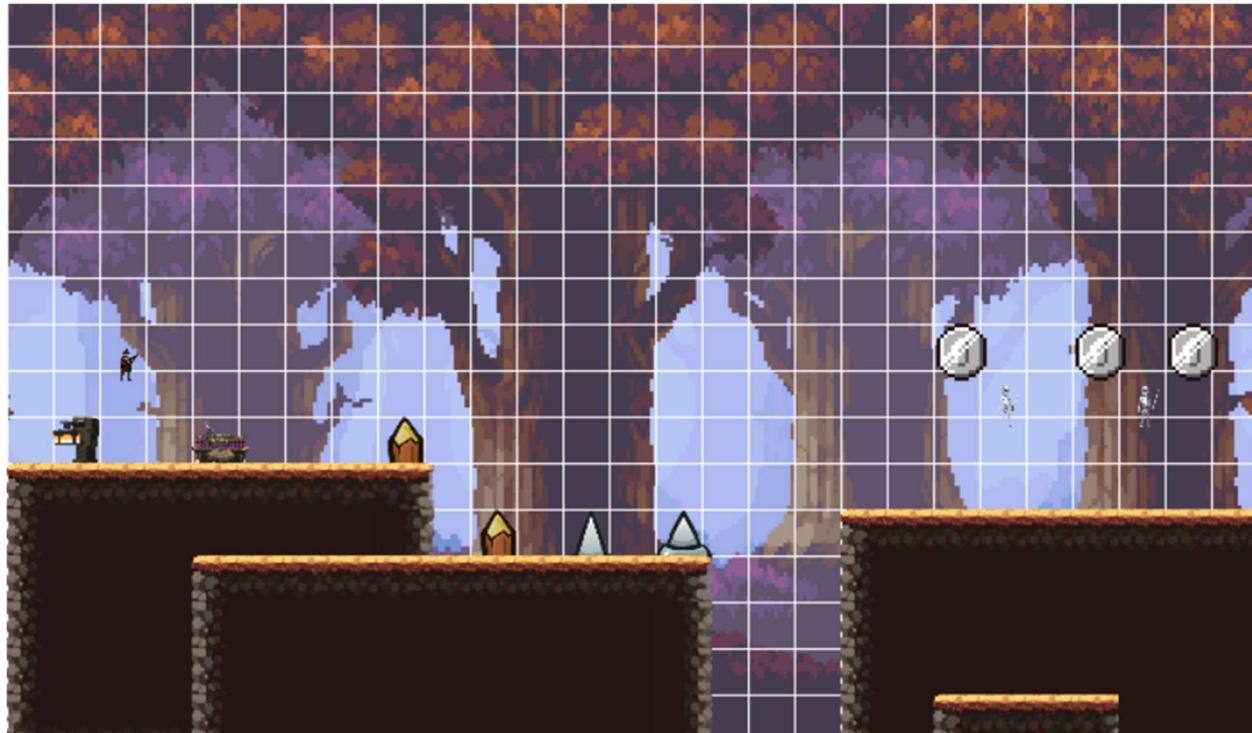
Tiếp theo là hàm vẽ dữ bản đồ thế giới lên màn hình

```
875     # Vẽ thế giới vào trong chỗ map
876     for y, row in enumerate(world_data):
877         for x, tile in enumerate(row):
878             if tile >= 0:
879                 surface.blit(img_list[tile], (x * TILE_SIZE_MAP - scroll, y * TILE_SIZE_MAP ))
```

Hàm sẽ đọc từng dòng, từng cột trong mảng `world_data` để vẽ lên bản đồ

- Hàm **draw\_world()** duyệt qua mỗi ô trong **world\_data** và vẽ hình ảnh tương ứng của ô lên màn hình.
- Biến **tile** là chỉ số của hình ảnh trong **img\_list**. Nếu giá trị của ô là -1, nghĩa là ô đó trống và không cần vẽ.

Kết quả của 3 hàm trên sẽ được như bên dưới



Tiếp theo là dùng hàm để vẽ các nút dùng để tạo level

```

881     # Nút load map
882     if load_btn.draw(screen):
883         scroll = 0
884     with open(f'Level/level{level}_data.csv', newline = '') as csvfile:
885         reader = csv.reader(csvfile, delimiter = ',')
886         for x, row in enumerate(reader):
887             for y, tile in enumerate(row):
888                 world_data[x][y] = int(tile)
889     # Nút lưu map
890     if save_btn.draw(screen):
891         with open(f'Level/level{level}_data.csv', 'w', newline = '') as csvfile:
892             writer = csv.writer(csvfile, delimiter = ',')
893             for row in world_data:
894                 writer.writerow(row)

```

- Nút **load\_btn** có nhiệm vụ khi ấn sẽ tải toàn bộ dữ liệu bản đồ theo level hiện tại lên màn hình cửa sổ

- Nút **save\_btn** có nhiệm vụ khi ấn vào nó sẽ lưu toàn bộ dữ liệu thế giới hiện tại vào file csv của level hiện tại trong thư mục Level

```

895     # Vẽ mảng các nút bên trái màn hình
896     button_count = 0
897     for button_count, i in enumerate(button_list):
898         if i.draw(screen):
899             current_tile = button_count
900             # Tô đậm nút ấn hiện tại
901             pygame.draw.rect(screen, RED, button_list[current_tile].rect, 3)
902

```

- Vòng lặp duyệt qua từng hình ảnh trong **img\_list** và tạo một nút (**tile\_button**) cho mỗi hình ảnh.
- Mỗi nút được vẽ tại vị trí dựa trên **SCREEN\_WIDTH**, **button\_col**, và **button\_row** ở lúc cài đặt mảng button ban đầu
- Ô được chọn được đánh dấu bằng cách vẽ một khung đỏ xung quanh nó.

```

858     # Chỗ này dùng để vẽ chữ nè
859     draw_text(f'Current Level: {level}', font, WHITE, 10, HEIGHT_MAP + 5)
860     draw_text(f'Current Tile: {current_tile}', font, WHITE, 10, HEIGHT_MAP + 20)
861     draw_text('Press UP/DOWN to change level', font, WHITE, 10, HEIGHT_MAP + 35)
862     draw_text('Press LEFT/RIGHT to scroll map', font, WHITE, 10, HEIGHT_MAP + 50)
863     draw_text('Press ESC to return to main menu', font, BLACK, 10, HEIGHT_MAP + 65)

```

Dùng lại hàm **draw\_text()** để vẽ thông tin hiện tại của bản đồ đang chỉnh sửa

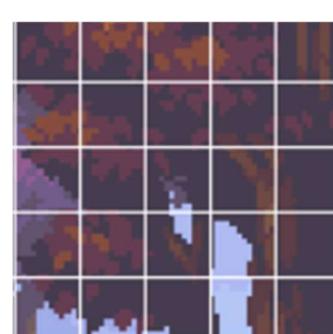


*Kết quả vẽ được công cụ để chỉnh sửa bản đồ*

## Xử lý các hành động của chuột và di chuyển level

```
903     # Di chuyển bản đồ
904     if scroll_left == True and scroll > 0:
905         scroll -= 5 * scroll_speed
906     if scroll_right == True and scroll < (COLUMNS * TILE_SIZE_MAP) - WIDTH_MAP:
907         scroll += 5 * scroll_speed
908
909     pos = pygame.mouse.get_pos()
910     x = (pos[0] + scroll) // TILE_SIZE_MAP
911     y = pos[1] // TILE_SIZE_MAP
912     # Kiểm tra xem chuột có nằm trong vùng vẽ không
913     if pos[0] < (704) and pos[1] < (416):
914         # Cập nhật tile vào world data
915         if pygame.mouse.get_pressed()[0] == 1:
916             if world_data[y][x] != current_tile:
917                 world_data[y][x] = current_tile
918             if pygame.mouse.get_pressed()[2] == 1:
919                 world_data[y][x] = -1
920
```

- Nếu biến **scroll\_left** được đặt thành **True** và bản đồ vẫn còn di chuyển về bên trái, thì biến **scroll** sẽ giảm đi **5 \* scroll\_speed**
- Tương tự, nếu biến **scroll\_right** được đặt thành **True** và bản đồ vẫn còn di chuyển về bên phải, thì biến **scroll** sẽ tăng lên **5 \* scroll\_speed**
- Tọa độ của con trỏ chuột được lấy bằng **pygame.mouse.get\_pos()**
- Dựa trên vị trí của con trỏ chuột và giá trị của biến **scroll**, chúng ta xác định được vị trí của ô trên bản đồ
- Nếu con trỏ chuột nằm trong khu vực của các bản đồ (**pos[0] < 704 and pos[1] < 416**):
  - Nếu nút trái chuột được nhấn, giá trị của ô sẽ được cập nhật thành giá trị của **current\_tile**, tức sẽ đặt ô đó lên màn hình.
  - Nếu nút phải chuột được nhấn, giá trị của ô sẽ được đặt lại thành **-1**, tức là ô trống, tức sẽ xoá ô đó đi

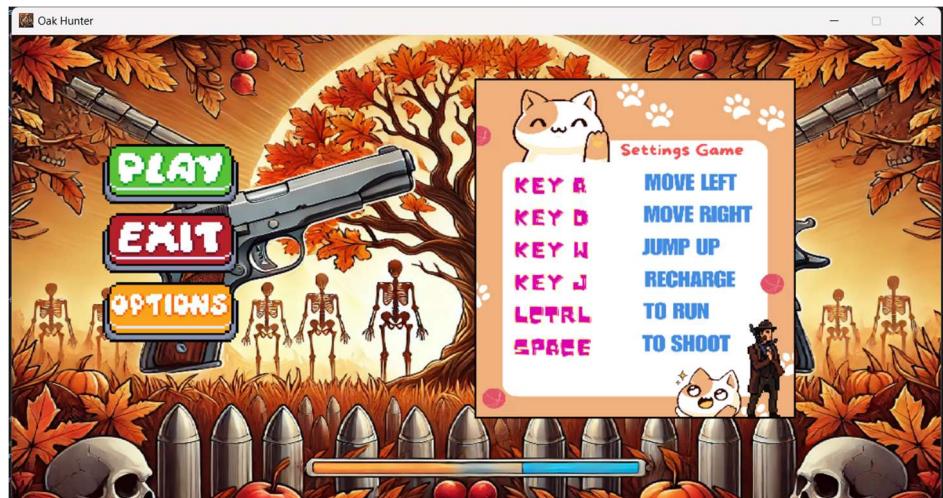


Cuối cùng là xử lý việc chuyển trái và chuyển phải bản đồ và thay đổi số level để tạo map

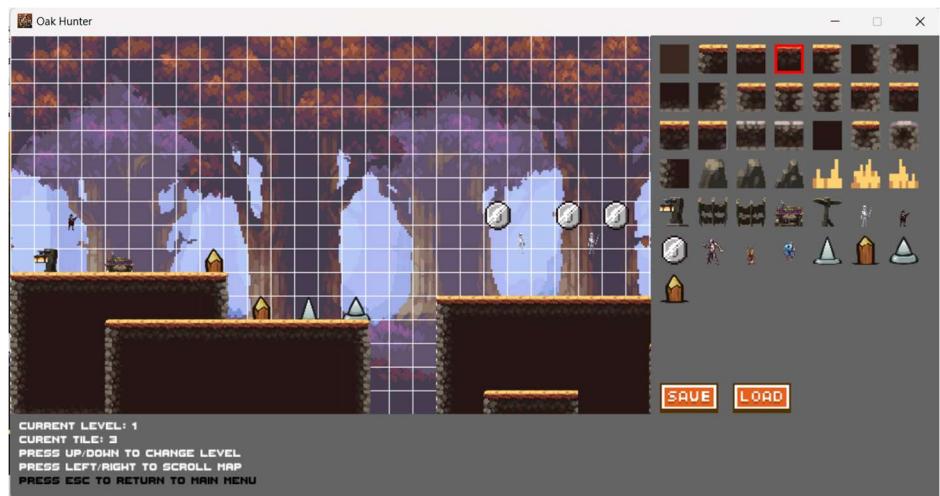
```
921     for event in pygame.event.get():
922         if event.type == pygame.KEYDOWN:
923             if event.key == pygame.K_LEFT:
924                 scroll_left = True
925             if event.key == pygame.K_RIGHT:
926                 scroll_right = True
927             if event.key == pygame.K_UP and level < MAX_LEVELS:
928                 level += 1
929             if event.key == pygame.K_DOWN and level > 1:
930                 level -= 1
931         if event.type == pygame.KEYUP:
932             if event.key == pygame.K_LEFT:
933                 scroll_left = False
934             if event.key == pygame.K_RIGHT:
935                 scroll_right = False
936         # Exit map editor
937         if event.type == pygame.KEYDOWN:
938             if event.key == pygame.K_ESCAPE:
939                 action_edit = False
940                 # Cập nhật lại dữ liệu thế giới
941                 world = reset_level()
942                 reset_data()
943                 world = World()
944                 player = world.process_data(world_data)
945                 home_game = True
946             # Exit pygame
947             if event.type == pygame.QUIT:
948                 pygame.quit()
949         # Update to screen
950         screen.blit(surface, (0, 0))
951
952     return action_edit, home_game
953
```

- Vòng lặp này xử lý các sự kiện từ bàn phím và cửa sổ
- Nếu người dùng nhấn nút ESC, biến **action\_edit** được đặt thành **False**, và biến **home\_game** sẽ được đặt về **True** và trả về 2 biến này ở cuối vòng lặp
- Vẽ lại bề mặt phụ lên lại bề mặt chính của cửa sổ bằng **screen.blit(surface, (0, 0))**
- Khi người dùng nhấn phím:
  - Nếu là phím mũi tên lên (**pygame.K\_UP**), cấp độ sẽ tăng lên 1.
  - Nếu là phím mũi tên xuống (**pygame.K\_DOWN**) và cấp độ lớn hơn 1, cấp độ sẽ giảm đi 1
  - Nếu là phím mũi tên trái (**pygame.K\_LEFT**), biến **scroll\_left** sẽ được đặt thành **True**
  - Nếu là phím mũi tên phải (**pygame.K\_RIGHT**), biến **scroll\_right** sẽ được đặt thành **True**

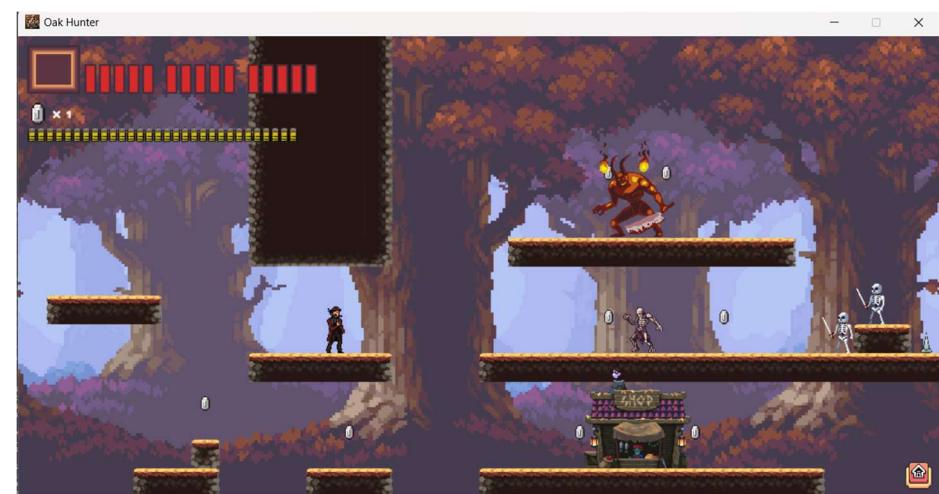
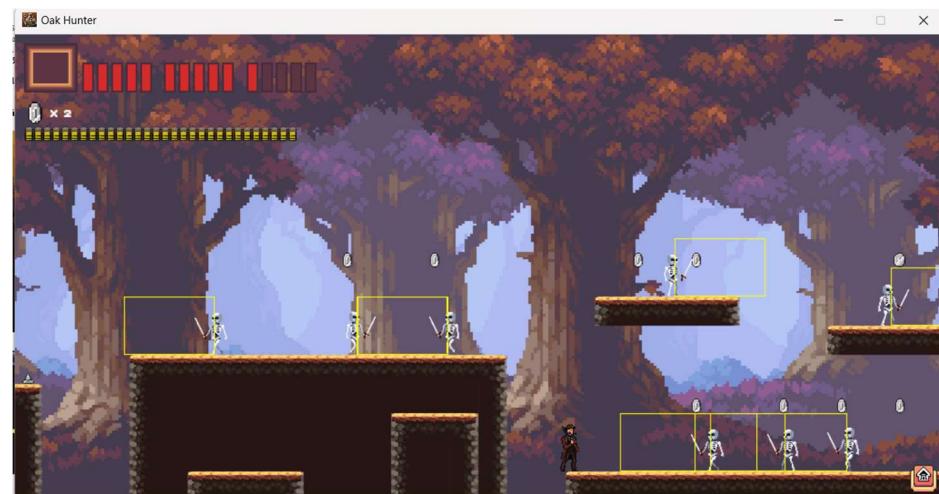
### 3.17 Một vài hình ảnh của trò chơi



Hình ảnh menu game



Hình ảnh chỉnh sửa map



Các hình ảnh của một vài bản đồ trong game

#### **4. Kết thúc và kết luận**

Sau một thời gian tìm hiểu đồ án, thu thập các kiến thức liên quan và tham khảo cách làm một số nơi trên Internet, em đã hoàn thành đồ án game Oak Hunter. Mặc dù rất cố gắng, nhưng vẫn không tránh khỏi thiếu sót và hạn chế. Em rất mong có được những ý kiến đánh giá, đóng góp của thầy để đồ án thêm hoàn thiện.

## **PHẦN III**

### **TÀI LIỆU THAM KHẢO**

- Tài liệu của Pygame: [Pygame Front Page — pygame v2.6.0 documentation](#)
- Loạt video hướng dẫn cách làm 1 game bắn súng của youtuber “Coding With Russ”: [PyGame - Shooter Game - YouTube](#)

