

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

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN MÔN NGÔN NGỮ LẬP  
TRÌNH PYTHON**

**Tên đề tài: Xây dựng ứng dụng game Dungeon World**

Sinh viên thực hiện:

**Nguyễn Khắc Tiệp - 3122410412**

**Ngô Quang Trường - 3122410440**

Mã Lớp: **DCT1204**

Giảng viên: **Trịnh Tấn Đạt**

Thành phố Hồ Chí Minh, 7 tháng 5 năm 2024

[illegible]

## **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 cảm ơn thầy Trịnh Tấn Đạt giáo viên giảng dạy đã nhiệt tình hướng dẫn chúng em hoàn thành đồ án lần này.

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

Em xin chân thành cảm ơn thầy ạ!

# MỤC LỤC

<b>Phần I. MỞ ĐẦU .....</b>	
<b>1. Giới thiệu đề tài .....</b>	<b>5</b>
<b>2. Lý do chọn đề tài.....</b>	<b>6</b>
<b>3. Mục đích - mục tiêu của đề tài.....</b>	<b>7</b>
<b>PHẦN II. NỘI DUNG .....</b>	
<b>1. Đôi nét về game Dungeon World .....</b>	
<b>1.1 Giới thiệu về Dungeon World.....</b>	<b>8</b>
<b>1.2 Mục tiêu của game Dungeon World.....</b>	<b>8</b>
<b>2. Yêu cầu đối với đồ án .....</b>	<b>8</b>
<b>3. Tiến hành xây dựng ứng dụng .....</b>	
<b>3.1. Thiết lập các giá trị ban đầu trong file config.py.....</b>	<b>8</b>
3.1.1 Thiết lập chiều rộng và chiều cao của cửa sổ game.....	8
3.1.2 Thiết lập kích thước camera, bản đồ và boss.....	8
3.1.3 Thiết lập thứ tự hiển thị các phần tử trong trò chơi.....	9
3.1.4 Thiết lập tốc độ người chơi, kẻ thù và đạn.....	9
3.1.5 Thiết lập khoảng cách giữa các lần bắn và phạm vi tấn công.....	10
3.1.6 Thiết lập các giá trị màu sắc.....	11
3.1.7 Thiết lập giá trị FPS của trò chơi.....	11
3.1.8 Thiết lập tỉ lệ xuất hiện của các vũ khí cho kẻ thù.....	11
3.1.9 Thiết lập bản đồ.....	11
<b>3.2 Xây dựng các lớp trong file sprites.py.....</b>	
3.2.1 Khai báo thư viện.....	16
3.2.2 Xây dựng lớp Spritesheet.....	16
3.2.3 Xây dựng Class Player: .....	17
3.2.4 Xây dựng Class Enemy: .....	25
3.2.5 Xây dựng Class Boss:.....	33
3.2.6 Xây dựng Class Block:.....	34
3.2.7 Xây dựng Class Ground:.....	35
3.2.8 Xây dựng Class Attack: :.....	36
3.2.9 Xây dựng Class Bullet: :.....	38
3.2.10 Xây dựng Class Gun: :.....	39
3.2.11 Xây dựng Class Glock, AK47, SNIPER: :.....	41
3.2.12 Xây dựng Class Button: :.....	44
3.2.13 Xây dựng Class MyMap: :.....	45
3.2.14 Xây dựng Class MapList: :.....	49
3.2.15 Xây dựng Class PlayerBar: :.....	51
3.2.16 Xây dựng Class BossHPBar: :.....	54
3.2.17 Xây dựng Class Torch: :.....	55
<b>3.3 Xây dựng File khởi tạo trò chơi:.....</b>	<b>56</b>
<b>4. Kết luận.....</b>	<b>65</b>

## Phần I. MỞ ĐẦU

### 1. Giới thiệu đề tài

Tên đề tài:

Xây dựng ứng dụng trò chơi Dungeon World với thư viện Pygame bằng ngôn ngữ lập trình Python.

### 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 ra đời và trở nên huyền thoại một thời. Cho nên em đã quyết định sử dụng thư viện Pygame của Python để xây dựng lại một game rất ấn tượng với em đó là Dungeon World.

### **3. Mục đích - mục tiêu của đề tài.**

#### **- Mục đích:**

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

#### **- Mục tiêu:**

- + Vận dụng được tính chất của lập trình hướng đối tượng.
- + Sử dụng thư viện Pygame vào việc xây dựng game Dungeon World

## PHẦN II. NỘI DUNG

### 1. Đôi nét về game Dungeon World

#### 1.1 Giới thiệu về Dungeon World

Dungeon World là trò chơi điện tử được lấy cảm hứng từ tựa game Soul Knife do nhóm sinh viên trường đại học Sài Gòn phát triển. Trò chơi được trình bày theo phong cách pixel, trong đó người chơi điều khiển một nhân vật, cố gắng tiêu diệt kẻ thù và boss mà không bị kẻ thù tiêu diệt. Người chơi sử dụng các phím A,D,S,W( hoặc cái phím mũi tên trái, phải, xuống, lên) để di chuyển nhân vật và sử dụng phím 1,2,3 để đổi vũ khí. Người chơi lần lượt đi qua các căn phòng và sẽ gặp kẻ thù, ở mỗi phòng sẽ có 2 giai đoạn( riêng phòng chứa boss thì chỉ có 1 giai đoạn mà thôi), khi người chơi tiêu diệt hết kẻ thù được tạo sẵn trong phòng đó thì sẽ tiếp tục tới giai đoạn 2 với kẻ thù số lượng ngẫu nhiên, khi tiêu diệt hết kẻ thù ở cả 2 giai đoạn thì qua phòng mới, lặp lại như vậy cho tới khi tất cả kẻ thù( bao gồm cả boss) ở tất cả các phòng bị tiêu diệt

#### 1.2 Mục tiêu của game Dungeon World

Mục tiêu của Dungeon World là tiêu diệt hết tất cả kẻ thù và boss để giành chiến thắng, nếu người chơi hết máu thì phải chơi lại từ đầu

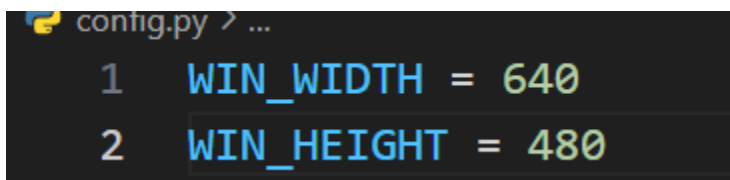
### 2. Yêu cầu đối với đồ án

Xây dựng ứng dụng Dungeon World dựa vào những kiến thức đã học về ngôn ngữ lập trình Python.

### 3. Tiến hành xây dựng ứng dụng

#### 3.1 Thiết lập các giá trị ban đầu trong file config.py

##### 3.1.1 Thiết lập chiều rộng và chiều cao của cửa sổ game



```
config.py > ...  
1  WIN_WIDTH = 640  
2  WIN_HEIGHT = 480
```

win\_width là chiều rộng của cửa sổ game, win\_height là chiều cao của cửa sổ game

##### 3.1.2 Thiết lập kích thước camera, bản đồ và boss

```
3  CAMERA_SIZE = 16
4  TILE_SIZE = 32
5  BOSS_SIZE=64
6
```

CAMERA\_SIZE là kích thước của camera, TILE\_SIZE là kích thước bản đồ, BOSS\_SIZE là kích thước của boss

### 3.1.3 Thiết lập thứ tự hiển thị các phần tử trong trò chơi

```
6
7
8  PLAYER_LAYER = 3
9  BLOCK_LAYER = 2
10 GROUND_LAYER = 1
11 ENERMY_LAYER = 4
12 BULLET_LAYER = 5
13 GUN_LAYER = 6
14 MAP_LAYER = 0
15 UI_LAYER = 10
16
```

Các giá trị trên được sử dụng để kiểm soát thứ tự hiển thị của các phần tử trong trò chơi, giúp quản lý hiển thị một cách dễ dàng và linh hoạt hơn.

### 3.1.4 Thiết lập tốc độ người chơi, kẻ thù và đạn



```

15  GLOCK_BULLET_SPEED = 10
16
17  PLAYER_SPEED = 5
18  ENEMY_SPEED = 2
19  GLOCK_BULLET_SPEED = 10
20  AK47_BULLET_SPEED = 12
21  SNIPER_BULLET_SPEED = 20
22

```

PLAYER\_SPEED là tốc độ của người chơi, ENEMY\_SPEED là tốc độ của kẻ thù, GLOCK\_BULLET\_SPEED, AK47\_BULLET\_SPEED và SNIPER\_BULLET\_SPEED là tốc độ của đạn

### 3.1.5 Thiết lập khoảng cách giữa các lần bắn và phạm vi tấn công

```

21  SNIPER_BULLET_SPEED = 20
22
23  #chỉnh lại khoảng thời gian giữa các lần bắn
24  ENEMY_SCOPE = 360
25  PLAYER_GLOCK_DELAY = 240
26  GLOCK_SCOPE = 160
27  WEAPON_SIZE = 42
28  PLAYER_AK47_DELAY = 100
29  AK47_SCOPE = 200
30  PLAYER_SNIPER_DELAY = 700
31  SNIPER_SCOPE = 300
32

```

Với ENEMY\_SCOPE, GLOCK\_SCOPE, AK47\_SCOPE, SNIPER\_SCOPE là phạm vi tấn công của kẻ thù và súng.

PLAYER\_GLOCK\_DELAY, PLAYER\_AK47\_DELAY, PLAYER\_SNIPER\_DELAY là khoảng cách giữa các lần bắn ở vũ khí của người chơi.  
WEAPON\_SIZE là kích thước của vũ khí, ở đây bao gồm glock,ak47 và sniper.

### 3.1.5 Thiết lập giá trị sát thương

```
32  
33  PLAYER_GLOCK_DMG = 1  
34  PLAYER_AK47_DMG = 2  
35  PLAYER_SNIPER_DMG = 3  
36
```

Với glock,ak47,sniper có sát thương lần lượt là 1,2,3.

### 3.1.6 Thiết lập các giá trị màu sắc

```
36  
37  RED = (255, 0, 0)  
38  BLACK = (0, 0, 0)  
39  BLUE = (0, 0, 255)  
40  WHITE = (255, 255, 255)  
41  YELLOW = (255, 255, 0)  
42  GREEN = (0, 255, 0)  
43  GREY = (128, 128, 128)  
44  BROWN = (255,160, 85)  
45  DARK_BROWN = (101,67,33)  
46
```

### 3.1.7 Thiết lập giá trị FPS của trò chơi

```
46
```

```
47  FPS = 60
```

```
48
```

### 3.1.8 Thiết lập tỉ lệ xuất hiện của các vũ khí cho kẻ thù

```
48
```

```
49  ENEMY_WEAPON_RATIO = {"glock": 0.5, "ak47": 0.3, "sniper": 0.2}
```

```
50
```

với tỉ lệ glock là 50%, ak47 là 30%, sniper là 20%

### 3.1.9 Thiết lập bản đồ

```

50
51 hpipemap = [
52     ' ',
53     ' ',
54     ' ',
55     ' ',
56     'BBBBBBBBBBBBBBBBBBBB',
57     '.....',
58     '.....',
59     '.....',
60     '.....',
61     '.....',
62     'BBBBBBBBBBBBBBBBBBBB',
63     ' ',
64     ' ',
65     ' ',
66     ' '
67 ]
68

```

hpipe<sub>map</sub> là phần sẽ nối theo chiều ngang 2 căn phòng lại với nhau

```
68
69 vpipemap = [
70     '      B.....B',
71     '      B.....B',
72     '      B.....B',
73     '      B.....B',
74     '      B.....B',
75     '      B.....B',
76     '      B.....B',
77     '      B.....B',
78     '      B.....B',
79     '      B.....B',
80     '      B.....B',
81     '      B.....B',
82     '      B.....B',
83     '      B.....B',
84     '      B.....B'
85 ]
86
```

vpipemap là phần sẽ nối theo chiều dọc 2 căn phòng

```

87 tilemaps = [
88     [
89         'BBBBBBBBBBBBBBBBBBBB',
90         'B.....B',
91         'B.....B',
92         'B.....B',
93         'B.....B',
94         'B.....B',
95         'B.....B',
96         'B.....P.....B',
97         'B.....B',
98         'B.....B',
99         'B.....B',
100        'B.....B',
101        'B.....B',
102        'B.....B',
103        'BBBBBBBBBBBBBBBBBBBB'
104    ],
105    [
106        'BBBBBBBBBBBBBBBBBBBB',
107        'B.....B',
108        'B.....B',
109        'B.....B',
110        'B.....B',
111        'B.....B',
112        'B.....B',
113        'B.....5.....B',
114        'B.....B',
115        'B.....B',
116        'B.....B',
117        'B.....B',
118        'B.....B',
119        'B.....B',
120        'BBBBBBBBBBBBBBBBBBBB'
121    ],

```

```

122  [
123      'BBBBBBBBBBBBBBBBBBBB',
124      'B.....B',
125      'B...BBB....E...B',
126      'B....B.....B',
127      'B....B.....E...B',
128      'B.....B.....B',
129      'B.....BBB.....B',
130      'B.....BB.....B',
131      'B...E....B.B.....B',
132      'B.....B',
133      'B.....B..B..B',
134      'B.....E....B..B..B',
135      'B.....BBBB..B',
136      'B.....B',
137      'BBBBBBBBBBBBBBBBBBBB'
138  ],
139  [
140      'BBBBBBBBBBBBBBBBBBBB',
141      'B.....B',
142      'B....B....B....B',
143      'B....B....BE...B',
144      'B...BBB.....BBB...B',
145      'B.....B',
146      'B.....B',
147      'B.....EE.....B',
148      'B.....B',
149      'B.....B',
150      'B...BBB.....BBB...B',
151      'B....B....B....B',
152      'B....B....B....B',
153      'B.....E.E..B',
154      'BBBBBBBBBBBBBBBBBBBB'
155  ],

```

```

156 [
157     'BBBBBBBBBBBBBBBBBBBB',
158     'B.....B',
159     'B.....E.....B',
160     'B....B....B....B',
161     'B....B....B....B',
162     'B...E...B..B..E...B',
163     'B.....BB.....B',
164     'B.....BB.....B',
165     'B...E...B..B..E...B',
166     'B....B....B....B',
167     'B....B....B....B',
168     'B.....E.....B',
169     'B.....B',
170     'B.....B',
171     'BBBBBBBBBBBBBBBBBBBB'
172 ],
173 [
174     'BBBBBBBBBBBBBBBBBBBB',
175     'B.....B',
176     'B...BBB.....B',
177     'B...B B.....E.....B',
178     'B...BBB.....B',
179     'B.....E.....B',
180     'B.....B..B.....B',
181     'B.....BB..B..E...B',
182     'B.....B.BB.....B',
183     'B.....B..B..E...B',
184     'B......BBBB..B',
185     'B.....E...B....B',
186     'B.....B..B..B',
187     'B......BBBB..B',
188     'BBBBBBBBBBBBBBBBBBBB'
189 ]
190 ]
191

```

tilemaps sẽ là bản đồ của game, với mỗi chữ B sẽ tượng trưng cho phần block, E sẽ là kẻ thù, mỗi dấu “.” sẽ là background, chữ P tượng trưng cho người chơi và số 5 tượng trưng cho boss.

Như vậy là việc thiết lập các giá trị ban đầu của game đã xong.

## 3.2 Xây dựng các lớp trong file sprites.py

### 3.2.1 Khai báo thư viện

```

1  import pygame
2  from config import *
3  import math
4  import random

```

### 3.2.2 Xây dựng lớp Spritesheet



```

6
7 class Spritesheet:
8     def __init__(self, filename):
9         self.sheet = pygame.image.load(filename).convert()
10
11     def get_sprite(self, x, y, width, height):
12         sprite = pygame.Surface((width, height))
13         sprite.blit(self.sheet, (0, 0), (x, y, width, height))
14         sprite.set_colorkey(BLACK)
15         return sprite
16

```

Đoạn code trên định nghĩa một lớp Python có tên là Spritesheet, được sử dụng để xử lý hình ảnh từ một tệp tin (file) hình ảnh cụ thể

Với **def \_\_init\_\_(self, filename):**  
**self.sheet = pygame.image.load(filename).convert()**

Phương thức khởi tạo, nhận vào một tham số filename, là đường dẫn đến tệp tin hình ảnh. Phương thức này tải tệp tin hình ảnh bằng thư viện Pygame và chuyển đổi nó thành một bề mặt.

**“def get\_sprite(self, x, y, width, height):**  
**sprite = pygame.Surface((width, height))**  
**sprite.blit(self.sheet, (0, 0), (x, y, width, height))**  
**sprite.set\_colorkey(BLACK)**  
**return sprite”**

Phương thức này nhận vào bốn tham số là x, y, width, và height, đại diện cho vị trí(x,y) và kích thước(width,height) của sprite trong hình ảnh.

**sprite = pygame.Surface((width, height)):** Tạo một đối tượng Surface mới trong Pygame. Surface là một đối tượng có thể vẽ lên, và ở đây nó sẽ được sử dụng để chứa sprite cắt ra.

**sprite.blit(self.sheet, (0, 0), (x, y, width, height)):** Phương thức blit được sử dụng để chèn nội dung từ một Surface này sang một Surface khác. Trong trường hợp này, chèn một phần của hình ảnh từ Spritesheet bắt đầu từ tọa độ (x, y) với kích thước (width, height), và sao chép nó vào bề mặt sprite từ vị trí (0, 0) của sprite. Sau đó, nó đặt màu chủ đề (colorkey) của sprite để loại bỏ phần màu nền và trả về sprite đã xử lý.

### 3.2.3 Xây dựng Class Player:

```

17 class Player(pygame.sprite.Sprite):
18     def __init__(self, game, x, y, mapx, mapy):
19         self.game = game
20         # game.player = self
21         self._layer = PLAYER_LAYER
22         self.groups = self.game.all_sprites
23         pygame.sprite.Sprite.__init__(self, self.groups)
24
25         self.x = mapx + x * TILE_SIZE
26         self.y = mapy + y * TILE_SIZE
27         self.width = TILE_SIZE
28         self.height = TILE_SIZE
29
30         self.x_change = 0
31         self.y_change = 0
32         self.facing = "right"
33         self.animation_loop = 1
34         self.attack_loop = 0
35         self.attacking = False
36
37         self.image = self.game.character_spritesheet.get_sprite(3, 2, self.width, self.height)
38
39         self.rect = self.image.get_rect()
40         self.rect.x = self.x
41         self.rect.y = self.y
42
43         self.right_animations = [self.game.character_spritesheet.get_sprite(3, 66, self.width, self.height),
44                                   self.game.character_spritesheet.get_sprite(35, 66, self.width, self.height),
45                                   self.game.character_spritesheet.get_sprite(67, 66, self.width, self.height)]
46
47         self.left_animations = [self.game.character_spritesheet.get_sprite(3, 98, self.width, self.height),
48                                  self.game.character_spritesheet.get_sprite(35, 98, self.width, self.height),
49                                  self.game.character_spritesheet.get_sprite(67, 98, self.width, self.height)]
50

```

```

51         self.rad = 0
52         self.score = 0
53
54         self.max_hp = 5
55         self.max_armour = 4
56         self.max_mana = 200
57
58         self.HP = self.max_hp
59         self.armour = self.max_armour
60         self.mana = self.max_mana
61
62         self.timer_hit = 0
63         self.timer_armour = 0
64         self.timer_attack = 0
65         self.weapons = []
66         self.weapon = None
67

```

1. `def __init__(self, game, x, y, mapx, mapy)::` Định nghĩa hàm khởi tạo của lớp `Player` với các tham số `game, x, y, mapx, mapy`.
2. `self.game = game:` Gán giá trị của tham số `game` vào thuộc tính `game` của đối tượng `Player`.
3. `self._layer = PLAYER_LAYER:` Gán giá trị của `PLAYER_LAYER` vào thuộc tính `_layer` của đối tượng `Player`. Có vẻ như đây là một giá trị cấp độ (layer) cho việc hiển thị đối tượng trong trò chơi.
4. `self.groups = self.game.all_sprites:` Gán giá trị của `self.game.all_sprites` vào thuộc tính `groups` của đối tượng `Player`. Có vẻ như đây là một cách để quản lý các nhóm đối tượng trong trò chơi.
5. `pygame.sprite.Sprite.__init__(self, self.groups):` Gọi hàm khởi tạo của lớp `Sprite` (một lớp trong thư viện `pygame`) với các nhóm được chỉ định, đảm bảo rằng đối tượng `Player` sẽ được thêm vào các nhóm này.
6. Các dòng tiếp theo đặt giá trị cho các thuộc tính như vị trí (`x, y`), kích thước (`width, height`), hình ảnh, các hình ảnh động, giá trị các biến như `HP, mana, armor`, vv.
7. `self.weapons = []:` Khởi tạo một danh sách rỗng để lưu trữ các vũ khí mà người chơi có thể sử dụng.
8. `self.weapon = None:` Khởi tạo biến `weapon` với giá trị `None`, tức là người chơi không sở hữu bất kỳ vũ khí nào ban đầu.

```
def set_weapons(self, weapons = None):
    if weapons == None:
        self.weapons = ["glock", "ak47", "sniper"]
        self.change_weapon(0)
    else:
        self.weapons = weapons
        self.change_weapon(0)
```

1. `def set_weapons(self, weapons = None)::` Định nghĩa phương thức `set_weapons` với tham số `weapons` mặc định là `None`.
2. `if weapons == None::` Kiểm tra xem tham số `weapons` có giá trị là `None` hay không.
  - Nếu `weapons` là `None`, tức là không có vũ khí nào được cung cấp, phương thức sẽ thực hiện các hành động sau:
    - Gán danh sách các vũ khí mặc định `["glock", "ak47", "sniper"]` vào thuộc tính `self.weapons`.
    - Gọi phương thức `change_weapon(0)` để chuyển đổi sang vũ khí đầu tiên trong danh sách.
  - Nếu `weapons` không phải là `None`, tức là có một danh sách vũ khí được cung cấp, phương thức sẽ thực hiện các hành động sau:

- Gán danh sách vũ khí được cung cấp vào thuộc tính **self.weapons**.
- Gọi phương thức **change\_weapon(0)** để chuyển đổi sang vũ khí đầu tiên trong danh sách mới.

```

76 def change_weapon(self, index):
77     if self.weapons[index] == None: return False
78     if self.weapons[index] == "glock":
79         if self.weapon: self.weapon.kill()
80         self.weapon = Glock(self.game, self)
81     if self.weapons[index] == "ak47":
82         if self.weapon: self.weapon.kill()
83         self.weapon = AK47(self.game, self)
84     if self.weapons[index] == "sniper":
85         if self.weapon: self.weapon.kill()
86         self.weapon = Sniper(self.game, self)
87
88 def update(self):
89     self.movement()
90     self.animate()
91     self.find_nearest_enemy()
92     self.collide_enemy()
93     self.collide_bullet()
94
95     self.rect.x += self.x_change
96     self.collide_blocks("x")
97     self.rect.y += self.y_change
98     self.collide_blocks("y")
99
100    self.x_change = 0
101    self.y_change = 0
102
103    #TODO: sau 4s không chiến đấu sẽ hồi 1 giáp / s
104    if self.armor < self.max_armor and pygame.time.get_ticks() - self.timer_attack > 3000:
105        if pygame.time.get_ticks() - self.timer_armor > 1000:
106            self.armor += 1
107            self.timer_armor = pygame.time.get_ticks()

```

#### 1. def change\_weapon(self, index):

- **index**: Chỉ số của vũ khí trong danh sách vũ khí mà người chơi muốn thay đổi.
- Kiểm tra xem vũ khí có tồn tại ở chỉ số **index** trong danh sách **weapons** hay không.
- Nếu không có vũ khí tại chỉ số đó (**None**), phương thức trả về **False** và không thực hiện thay đổi.
- Nếu có vũ khí tại chỉ số đó, phương thức thực hiện các hành động sau:
  - Nếu đã có vũ khí hiện tại, phương thức tiến hành loại bỏ vũ khí đó.
  - Tạo ra một vũ khí mới tương ứng với loại vũ khí ở chỉ số **index** và gán nó vào thuộc tính **weapon**.

#### 2. def update(self):

- Cập nhật trạng thái của người chơi trong mỗi frame của trò chơi.

- Thực hiện các hành động như di chuyển, animation, xác định kẻ thù gần nhất, va chạm với kẻ thù và đạn, xử lý va chạm với các khối trong môi trường.
- Reset các biến di chuyển (**self.x\_change** và **self.y\_change**) về 0 để chuẩn bị cho frame tiếp theo.
- Kiểm tra và cập nhật giá trị của giáp (**armour**) của người chơi sau một khoảng thời gian nhất định nếu giáp nhỏ hơn giá trị tối đa (**max\_armour**) và đã trôi qua một khoảng thời gian nhất định.

```

3. def movement(self):
4.     keys = pygame.key.get_pressed()
5.     if keys[pygame.K_LEFT] or keys[pygame.K_a]:
6.         if(self.rect.centerx < WIN_WIDTH/2 + CAMERA_SIZE):
7.             for sprite in self.game.all_sprites:
8.                 sprite.rect.x += PLAYER_SPEED
9.             for attack in self.game.attacks:
10.                attack.rect.x -= PLAYER_SPEED
11.            self.x_change = -PLAYER_SPEED
12.            if self.find_nearest_enemy() == None: self.facing = "left"
13.    if keys[pygame.K_RIGHT] or keys[pygame.K_d]:
14.        if(self.rect.centerx > WIN_WIDTH/2 - CAMERA_SIZE):
15.            for sprite in self.game.all_sprites:
16.                sprite.rect.x -= PLAYER_SPEED
17.            for attack in self.game.attacks:
18.                attack.rect.x += PLAYER_SPEED
19.            self.x_change = PLAYER_SPEED
20.            if self.find_nearest_enemy() == None: self.facing = "right"
21.    if keys[pygame.K_UP] or keys[pygame.K_w]:
22.        if(self.rect.centery < WIN_HEIGHT/2 + CAMERA_SIZE):
23.            for sprite in self.game.all_sprites:
24.                sprite.rect.y += PLAYER_SPEED
25.            for attack in self.game.attacks:
26.                attack.rect.y -= PLAYER_SPEED
27.            self.y_change = -PLAYER_SPEED
28.    if keys[pygame.K_DOWN] or keys[pygame.K_s]:
29.        if(self.rect.centery > WIN_HEIGHT/2 - CAMERA_SIZE):
30.            for sprite in self.game.all_sprites:
31.                sprite.rect.y -= PLAYER_SPEED
32.            for attack in self.game.attacks:
33.                attack.rect.y += PLAYER_SPEED
34.            self.y_change = PLAYER_SPEED
35.
36.    def collide_enemy(self):
37.        hits = pygame.sprite.spritecollide(self, self.game.enemies, False)
38.        if hits:
39.            self.take_dmg(1)

```

#### 1. def movement(self):

- Phương thức này xử lý việc di chuyển của người chơi dựa trên các phím mũi tên hoặc các phím WASD được nhấn.

- Nếu người chơi nhấn phím mũi tên trái hoặc A, người chơi sẽ di chuyển sang trái. Nếu vị trí của người chơi không vượt quá một khoảng cách quy định (được tính dựa trên kích thước của sổ trò chơi và camera), mọi sprite trong trò chơi và tất cả các tấn công cũng sẽ di chuyển sang phải để tạo hiệu ứng di chuyển của camera. Đồng thời, **self.x\_change** được gán bằng giá trị âm để di chuyển sang trái.
- Tương tự, nếu người chơi nhấn phím mũi tên phải hoặc D, người chơi sẽ di chuyển sang phải. Nếu vị trí của người chơi không vượt quá một khoảng cách quy định, mọi sprite trong trò chơi và tất cả các tấn công cũng sẽ di chuyển sang trái để tạo hiệu ứng di chuyển của camera. Đồng thời, **self.x\_change** được gán bằng giá trị dương để di chuyển sang phải.
- Tương tự cho việc di chuyển lên và xuống với phím mũi tên lên và xuống hoặc phím W và S.

## 2. **def collide\_enemy(self):**

- Phương thức này xác định va chạm giữa người chơi và các kẻ địch.
- Sử dụng hàm **pygame.sprite.spritecollide()** để kiểm tra va chạm giữa người chơi và nhóm các sprite địch (**self.game.enemies**). Kết quả của việc này là một danh sách các sprite mà người chơi va chạm, với tham số **false** thì khi va chạm sẽ không xóa sprite.
- Nếu có va chạm xảy ra (danh sách **hits** không rỗng), người chơi sẽ gán một đơn vị sát thương cho mình thông qua hàm **take\_dmg()**.

```

3. def take_dmg(self, dmg):
4.     if pygame.time.get_ticks() - self.timer_hit > 800:
5.         if(self.armor > 0):
6.             self.armor -= dmg
7.         else:
8.             self.HP -= dmg
9.             self.timer_attack = pygame.time.get_ticks()
10.            self.timer_hit = pygame.time.get_ticks()
11.            if(self.HP <= 0):
12.                self.game.playing = False
13.
14.    def collide_bullet(self):
15.        hits = pygame.sprite.spritecollide(self, self.game.enemies_bullets,
False)
16.        if hits:
17.            self.take_dmg(1) # buff undead
18.            hits[0].kill()
19.
20.    def collide_blocks(self, dir):
21.        hits = pygame.sprite.spritecollide(self, self.game.blocks, False)
22.        if hits:
23.            if (dir == "x"):
24.                if self.x_change > 0:

```

```

25.         self.rect.right = hits[0].rect.left
26.         if self.x_change < 0:
27.             self.rect.left = hits[0].rect.right
28.         if (dir == "y"):
29.             if self.y_change > 0:
30.                 self.rect.bottom = hits[0].rect.top
31.             if self.y_change < 0:
32.                 self.rect.top = hits[0].rect.bottom

```

### 1. **def take\_dmg(self, dmg):**

- Phương thức này xử lý việc người chơi nhận sát thương.
- Đầu tiên, kiểm tra xem thời gian kể từ lần va chạm trước đó đã vượt quá 800 miligiây hay chưa. Nếu chưa, không xử lý sát thương mới.
- Nếu có giá trị giáp (**armour**) lớn hơn 0, giảm giá trị giáp đi **dmg**, ngược lại giảm giá trị máu (**HP**) đi **dmg**.
- Gán giá trị của **timer\_attack** và **timer\_hit** bằng thời gian hiện tại, để đảm bảo rằng không có sát thương mới được nhận trong một khoảng thời gian ngắn sau khi nhận sát thương trước đó.
- Nếu máu của người chơi dưới hoặc bằng 0, đặt trạng thái trò chơi (**playing**) thành False để kết thúc trò chơi.

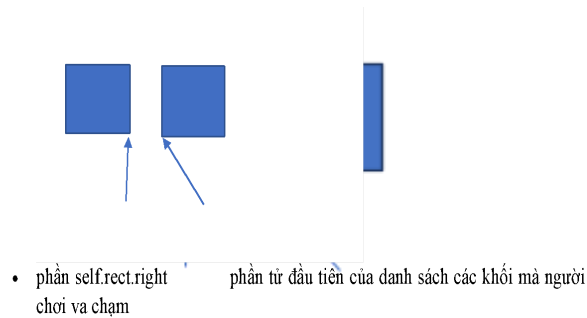
### 2. **def collide\_bullet(self):**

- Phương thức này xác định va chạm giữa người chơi và đạn của kẻ địch.
- Sử dụng hàm **pygame.sprite.spritecollide()** để kiểm tra va chạm giữa người chơi và nhóm các đạn của kẻ địch (**self.game.enemies\_bullets**). Kết quả của việc này là một danh sách các đạn mà người chơi va chạm.
- Nếu có va chạm xảy ra (danh sách **hits** không rỗng), gọi phương thức **take\_dmg()** với giá trị sát thương là 1 (hoặc bất kỳ giá trị sát thương nào khác). Đồng thời, loại bỏ đạn va chạm đầu tiên khỏi trò chơi.

### 3. **def collide\_blocks(self, dir):**

- Phương thức này xử lý va chạm giữa người chơi và các khối trong môi trường.
- Sử dụng hàm **pygame.sprite.spritecollide()** để kiểm tra va chạm giữa người chơi và nhóm các khối (**self.game.blocks**). Kết quả của việc này là một danh sách các khối mà người chơi va chạm.
- Nếu có va chạm xảy ra (danh sách **hits** không rỗng), kiểm tra hướng của va chạm (**dir** là "x" hoặc "y") và điều chỉnh vị trí của người chơi để ngăn cản di chuyển tiếp theo theo hướng đó.

- Ví dụ: ở đây khi di chuyển theo chiều ngang và đang đi về bên phải, điều này



đảm bảo việc người chơi sẽ không thể đi xuyên qua các khối block

```

4.         if (dir == "x"):
5.             if self.x_change > 0:
6.                 self.rect.right = hits[0].rect.left

7. def animate(self):
8.     if self.facing == "left":
9.         if self.x_change == 0 and self.y_change == 0:
10.            self.image = self.game.character_spritesheet.get_sprite(3,
11.            98, self.width, self.height)
12.        else:
13.            self.image =
14.            self.left_animations[math.floor(self.animation_loop)]
15.            self.animation_loop += 0.1
16.
17.     if self.facing == "right":
18.         if self.x_change == 0 and self.y_change == 0:
19.            self.image = self.game.character_spritesheet.get_sprite(3,
20.            66, self.width, self.height)
21.        else:
22.            self.image =
23.            self.right_animations[math.floor(self.animation_loop)]
24.            self.animation_loop += 0.1
25.
26.     if self.animation_loop >= 3:
27.         self.animation_loop = 1
28.
29. def find_nearest_enemy(self):
30.     #update rad to place the gun
31.     if self.y_change != 0 or self.x_change != 0:
32.         self.rad = math.atan2(self.y_change, self.x_change)
33.         nearest_enemy = None
34.         if(self.game.enemies.sprites()):
35.             nearest_enemy = min(self.game.enemies, key=lambda x:
36.             math.sqrt((x.rect.x - self.rect.x)**2 + (x.rect.y - self.rect.y)**2))
37.             if math.sqrt((nearest_enemy.rect.x - self.rect.x)**2 +
38.             (nearest_enemy.rect.y - self.rect.y)**2) < self.weapon.scope:

```



```

33.         dy = nearest_enemy.rect.y - self.rect.y
34.         dx = nearest_enemy.rect.x - self.rect.x
35.         self.rad = math.atan2(dy, dx)
36.         return nearest_enemy

```

#### 1. **def animate(self):**

- Phương thức này điều chỉnh hình ảnh của người chơi để tạo hiệu ứng hoạt hình khi người chơi di chuyển.
- Nếu người chơi đang hướng về bên trái và không di chuyển (**self.x\_change == 0** và **self.y\_change == 0**), hình ảnh sẽ được thiết lập để hiển thị hình ảnh chế độ đứng yên hướng về bên trái.
- Ngược lại, nếu người chơi đang hướng về bên trái và đang di chuyển, hình ảnh sẽ được thiết lập để hiển thị một trong các hình ảnh chuyển động hướng về bên trái, lấy từ danh sách **left\_animations**.
- Tương tự, nếu người chơi đang hướng về bên phải, hình ảnh sẽ được thiết lập tương ứng từ danh sách **right\_animations**.
- **self.animation\_loop** được tăng lên mỗi lần hàm được gọi để chuyển đổi giữa các hình ảnh chuyển động.

#### 2. **def find\_nearest\_enemy(self):**

- Phương thức này tìm kẻ địch gần nhất với người chơi.
- Trước tiên, cập nhật góc radian (**self.rad**) để đặt vị trí của súng.
- Nếu có di chuyển (**self.y\_change != 0** hoặc **self.x\_change != 0**), góc radian sẽ được tính toán dựa trên sự thay đổi vị trí của người chơi.
- Tìm kẻ địch gần nhất bằng cách sử dụng hàm **min()** với một hàm lambda để tính khoảng cách Euclide giữa người chơi và mỗi kẻ địch trong nhóm **enemies**. Kẻ địch gần nhất sẽ được trả về.
- Nếu khoảng cách giữa người chơi và kẻ địch gần nhất nhỏ hơn phạm vi của vũ khí (**self.weapon.scope**), góc radian sẽ được điều chỉnh để hướng về kẻ địch đó.
- Phương thức trả về kẻ địch gần nhất tìm thấy.

### 3.2.4 Xây dựng Class Enemy:

```

class Enemy(pygame.sprite.Sprite):
    def __init__(self, game, x, y, mapx, mapy, map, weapon_ratio_dict =
ENEMY_WEAPON_RATIO):
        self.game = game
        self._layer = ENEMY_LAYER
        self.groups = self.game.all_sprites, self.game.enemies
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.x = mapx + x * TILE_SIZE
        self.y = mapy + y * TILE_SIZE
        self.width = TILE_SIZE
        self.height = TILE_SIZE

        self.x_change = 0
        self.y_change = 0

        self.facing = "right"
        self.animation_loop = 1
        self.movement_loop = 0
        self.max_travel = random.randint(30, 60)

        self.image = self.game.enemy_spritesheet.get_sprite(3, 2, self.width,
self.height)
        self.image.set_colorkey(BLACK)

        self.rect = self.image.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y

        self.right_animations = [self.game.enemy_spritesheet.get_sprite(3, 66,
self.width, self.height),
                                self.game.enemy_spritesheet.get_sprite(35, 66,
self.width, self.height),
                                self.game.enemy_spritesheet.get_sprite(67, 66,
self.width, self.height)]

        self.left_animations = [self.game.enemy_spritesheet.get_sprite(3, 98,
self.width, self.height),
                                self.game.enemy_spritesheet.get_sprite(35, 98,
self.width, self.height),
                                self.game.enemy_spritesheet.get_sprite(67, 98,
self.width, self.height)]

        self.HP = 3
        self.room = map
        self.rad = 0

        rand_weapon = random.choices(list(weapon_ratio_dict.keys()),
weights=weapon_ratio_dict.values(), k=1)[0]
        if rand_weapon == "glock":
            self.weapon = Glock(self.game, self, PLAYER_GLOCK_DELAY)
        elif rand_weapon == "ak47":
            self.weapon = AK47(self.game, self, PLAYER_AK47_DELAY)

```

```

        else:
            self.weapon = Sniper(self.game, self, PLAYER_SNIPER_DELAY)

        self.attacking = False
        self.rand = random.randint(1, 4)

    def update(self):
        self.movement()
        self.collide_bullet()
        self.animate()
        self.rect.x += self.x_change
        self.collide_blocks("x")
        self.rect.y += self.y_change
        self.collide_blocks("y")

        self.x_change = 0
        self.y_change = 0

    def animate(self):
        if self.facing == "left":
            if self.x_change == 0 and self.y_change == 0:
                self.image = self.left_animations[0]
            else:
                self.image = self.left_animations[math.floor(self.animation_loop)]
                self.animation_loop += 0.1
        if self.facing == "right":
            if self.x_change == 0 and self.y_change == 0:
                self.image = self.right_animations[0]
            else:
                self.image = self.right_animations[math.floor(self.animation_loop)]
                self.animation_loop += 0.1

        if self.animation_loop >= 3:
            self.animation_loop = 1

    def collide_blocks(self, dir):
        hits = pygame.sprite.spritecollide(self, self.game.blocks, False)
        hits += pygame.sprite.spritecollide(self, self.game.entrances, False)
        if hits:
            if (dir == "x"):
                if self.x_change > 0:
                    self.rect.right = hits[0].rect.left
                if self.x_change < 0:
                    self.rect.left = hits[0].rect.right
            if (dir == "y"):
                if self.y_change > 0:
                    self.rect.bottom = hits[0].rect.top
                if self.y_change < 0:
                    self.rect.top = hits[0].rect.bottom

    def collide_bullet(self):
        hits = pygame.sprite.spritecollide(self, self.game.bullets, False)
        if hits:

```

```

        self.HP -= hits[0].dmg
        hits[0].kill()
        if self.HP <= 0:
            self.kill()
            if self.game.player.mana <= self.game.player.max_mana - 10:
                self.game.player.mana += 10
            else :
                self.game.player.mana = self.game.player.max_mana

    def movement(self):
        if self.y_change != 0 or self.x_change != 0:
            self.rad = math.atan2(self.y_change, self.x_change)
            player = self.game.player
            distance = math.sqrt((player.rect.centerx - self.rect.centerx)**2 +
(player.rect.centery - self.rect.centery)**2)
            if distance < self.weapon.scope + 96 and self.room.open == False:
                self.taunted_movement(distance)
            else:
                self.normal_movement()

    def normal_movement(self):
        if self.rand == 1:
            self.y_change -= ENEMY_SPEED
        if self.rand == 2:
            self.y_change += ENEMY_SPEED
        if self.rand == 3:
            self.x_change -= ENEMY_SPEED
            self.facing = "left"
            self.rad = math.pi
        if self.rand == 4:
            self.x_change += ENEMY_SPEED
            self.facing = "right"
            self.rad = 0

        self.movement_loop -= 1
        if self.movement_loop <= -self.max_travel:
            self.movement_loop = 0
            self.rand = random.randint(1, 4)
            self.max_travel = random.randint(30, 60)
        pass

    def taunted_movement(self, distance):
        player = self.game.player
        dy = player.rect.centery - self.rect.centery
        dx = player.rect.centerx - self.rect.centerx
        self.rad = math.atan2(dy, dx)
        if distance > self.weapon.scope:
            self.x_change += ENEMY_SPEED * math.cos(self.rad)
            self.y_change += ENEMY_SPEED * math.sin(self.rad)
        elif self.weapon.can_shoot():
            self.weapon.shoot()

#cu cach 1 khoang thoi gian, boss lai lam mot hanh dong ngau nhien

```

```
def kill(self):
    self.weapon.kill()
    pygame.sprite.Sprite.kill(self)
```

1. **\_\_init\_\_(self, game, x, y, mapx, mapy, map, weapon\_ratio\_dict = ENEMY\_WEAPON\_RATIO):**

- Phương thức khởi tạo của class Enemy, nhận các đối số như trò chơi (game), vị trí ban đầu (x và y), và các thông số khác như mapx, mapy, map, và weapon\_ratio\_dict (tỉ lệ xuất hiện của 1 trong 3 vũ khí đã được khai báo trong file config.py).
- self.game = game: Gán đối tượng trò chơi cho thuộc tính game.
- self.\_layer = ENEMY\_LAYER: Gán layer của sprite của Enemy bằng hằng số ENEMY\_LAYER.
- self.groups = self.game.all\_sprites, self.game.enemies: Gán self.groups bằng một tuple chứa các nhóm sprite mà sprite của Enemy thuộc về trong trò chơi.
- pygame.sprite.Sprite.\_\_init\_\_(self, self.groups): Gọi hàm khởi tạo của lớp cha pygame.sprite.Sprite để khởi tạo sprite.
- self.x = mapx + x \* TILE\_SIZE và self.y = mapy + y \* TILE\_SIZE: Tính toán vị trí thực tế của sprite trong trò chơi dựa trên vị trí ban đầu và kích thước của mỗi ô.
- self.width = TILE\_SIZE và self.height = TILE\_SIZE: Gán kích thước của sprite bằng kích thước của mỗi ô trên bản đồ.
- self.x\_change = 0 và self.y\_change = 0: Khởi tạo các biến để lưu trữ sự thay đổi vị trí của sprite.
- self.facing = "right": Đặt hướng ban đầu của sprite là "phải".
- self.animation\_loop = 1 và self.movement\_loop = 0: Khởi tạo các biến để quản lý hoạt hình và chuyển động của sprite.
- self.max\_travel = random.randint(30, 60): Đặt giới hạn tối đa cho khoảng cách di chuyển của kẻ thù.
- self.image = self.game.enemy\_spritesheet.get\_sprite(3, 2, self.width, self.height): Tạo hình ảnh cho sprite từ sprite sheet của Enemy, với tọa độ (3, 2) trên sprite sheet và kích thước là self.width và self.height.
- self.image.set\_colorkey(BLACK): Thiết lập màu chính là màu trong suốt để làm nền trong hình ảnh.
- self.rect = self.image.get\_rect(): Tạo một hình chữ nhật bao quanh sprite.
- self.rect.x = self.x và self.rect.y = self.y: Đặt vị trí ban đầu của sprite trong trò chơi.

- `self.right_animations = [...]`, `self.left_animations = [...]`: Tạo danh sách các hình ảnh cho hoạt ảnh khi sprite di chuyển sang phải và sang trái.
- `self.HP = 3`: Đặt điểm máu của Enemy là 3.
- `self.room = map`: Gán vị trí hiện tại của sprite trong bản đồ cho thuộc tính `room`.
- `self.rad = 0`: Khởi tạo góc xoay của sprite.
- `rand_weapon = random.choices(...)`: Chọn ngẫu nhiên một loại vũ khí dựa trên trọng số trong `weapon_ratio_dict`.
- `if rand_weapon == "glock": ... elif rand_weapon == "ak47": ... else: ...`: Tạo ra một loại vũ khí ngẫu nhiên cho Enemy dựa trên lựa chọn từ bước trên.
- `self.attacking = False`: Đặt trạng thái tấn công của sprite là False.
- `self.rand = random.randint(1, 4)`: Chọn ngẫu nhiên một giá trị từ 1 đến 4 và gán cho thuộc tính `rand`.

## 2. `update(self)`:

- Phương thức cập nhật trạng thái của kẻ địch trong mỗi frame của trò chơi.
- Xử lý di chuyển, va chạm với đạn, hoạt hình và va chạm với các khối trong môi trường.
- Reset các biến di chuyển (`self.x_change` và `self.y_change`) về 0 để chuẩn bị cho frame tiếp theo.

## 3. `animate(self)`:

- Phương thức này điều khiển việc hoạt hình của kẻ địch.
- Dựa vào hướng di chuyển của kẻ địch, chọn hình ảnh tương ứng từ các bộ hình ảnh hoạt hình.

Tạo hiệu ứng hoạt hình bằng cách thay đổi giá trị `self.animation_loop`.

```
if self.facing == "left":
    if self.x_change == 0 and self.y_change == 0:
        self.image = self.left_animations[0]
    else:
        self.image =
self.left_animations[math.floor(self.animation_loop)]
        self.animation_loop += 0.1
if self.animation_loop >= 3:
    self.animation_loop = 1
```

- Ở đây khi kẻ thù đang quay mặt về bên trái, nếu không di chuyển thì để hình ảnh của kẻ thù là phần tử đầu tiên trong `left_animations`. Nếu đang di chuyển thì gán hình ảnh của kẻ thù là một trong các hình ảnh trong danh sách hoạt ảnh tùy thuộc vào giá trị của `animation_loop`. Hàm `math.floor()` được sử dụng để

làm tròn giá trị `animation_loop` xuống số nguyên gần nhất. Bởi vì `left_animations` có 3 giá trị nên khi `animation_loop >= 3` thì ta đặt lại nó về 1, điều này đảm bảo cho hình ảnh kẻ thù sẽ luôn hiển thị ảnh trong phần `left_animation`.

#### 4. **collide\_blocks(self, dir):**

- Xử lý va chạm giữa kẻ địch và các khối trong môi trường.
- Nếu có va chạm xảy ra, điều chỉnh vị trí của kẻ địch để ngăn cản di chuyển tiếp theo theo hướng đó.
- Cũng tương tự như trong phần xử lý va chạm của player

#### 5. **collide\_bullet(self):**

- Xử lý va chạm giữa kẻ địch và đạn của người chơi.
- Giảm máu của kẻ địch khi va chạm với đạn. Nếu máu nhỏ hơn hoặc bằng 0, loại bỏ kẻ địch khỏi trò chơi.

#### 6. **movement(self):**

- Xử lý việc di chuyển của kẻ địch, bao gồm cả di chuyển bình thường và di chuyển dựa trên việc bị thách thức bởi người chơi.
- `if self.y_change != 0 or self.x_change != 0::` Kiểm tra xem kẻ thù đang di chuyển hay không (nếu `y_change` hoặc `x_change` khác 0) nếu đang di chuyển thì thực hiện đoạn code phía dưới.
- `self.rad = math.atan2(self.y_change, self.x_change):` Tính toán góc radian của vector di chuyển của kẻ thù dựa trên `y_change` và `x_change` sử dụng hàm `math.atan2()`.
- `player = self.game.player:` Lấy đối tượng của người chơi từ trò chơi.
- `distance = math.sqrt((player.rect.centerx - self.rect.centerx)**2 + (player.rect.centery - self.rect.centery)**2):` Tính toán khoảng cách giữa kẻ thù và người chơi bằng cách sử dụng công thức khoảng cách Euclide.
- `if distance < self.weapon.scope + 96 and self.room.open == False::` Kiểm tra xem người chơi có trong phạm vi tấn công của kẻ thù không và phòng đang đóng hay không.
- `self.taunted_movement(distance):` Nếu người chơi trong phạm vi tấn công và phòng đang đóng, thì gọi hàm `taunted_movement()` để xử lý chuyển động của kẻ thù.
- `else::` Nếu không, thì gọi hàm `normal_movement()` để xử lý chuyển động bình thường của kẻ thù.

#### 7. **normal\_movement(self):**

- Xử lý việc di chuyển bình thường của kẻ địch khi không bị thách thức bởi người chơi.
- `if self.rand == 1`: Kiểm tra xem giá trị của biến `rand` có bằng 1 không. Nếu có, kẻ thù sẽ di chuyển lên trên.
- `self.y_change -= ENEMY_SPEED`: Giảm giá trị của `y_change` để di chuyển kẻ thù lên trên theo hướng y.
- `if self.rand == 2`: Kiểm tra xem giá trị của biến `rand` có bằng 2 không. Nếu có, kẻ thù sẽ di chuyển xuống dưới.
- `self.y_change += ENEMY_SPEED`: Tăng giá trị của `y_change` để di chuyển kẻ thù xuống dưới theo hướng y.
- `if self.rand == 3`: Kiểm tra xem giá trị của biến `rand` có bằng 3 không. Nếu có, kẻ thù sẽ di chuyển sang trái.
- `self.x_change -= ENEMY_SPEED`: Giảm giá trị của `x_change` để di chuyển kẻ thù sang trái theo hướng x.
- `self.facing = "left"`: Đặt hướng của kẻ thù thành "trái".
- `self.rad = math.pi`: Đặt góc radian của kẻ thù thành  $\pi$  (pi), tương ứng với hướng sang trái.
- `if self.rand == 4`: Kiểm tra xem giá trị của biến `rand` có bằng 4 không. Nếu có, kẻ thù sẽ di chuyển sang phải.
- `self.x_change += ENEMY_SPEED`: Tăng giá trị của `x_change` để di chuyển kẻ thù sang phải theo hướng x.
- `self.facing = "right"`: Đặt hướng của kẻ thù thành "phải".
- `self.rad = 0`: Đặt góc radian của kẻ thù thành 0, tương ứng với hướng sang phải.
- `self.movement_loop -= 1`: Giảm giá trị của biến `movement_loop` đi 1.
- `if self.movement_loop <= -self.max_travel`: Kiểm tra xem biến `movement_loop` có nhỏ hơn hoặc bằng âm giá trị của `max_travel` không. Nếu có, tức là kẻ thù đã đi qua một số bước di chuyển cố định, ta cần đặt lại giá trị cho `movement_loop` và `rand` để tạo ra sự ngẫu nhiên trong hành vi di chuyển của kẻ thù.
- `self.movement_loop = 0`: Đặt lại giá trị của `movement_loop` về 0.
- `self.rand = random.randint(1, 4)`: Chọn một giá trị ngẫu nhiên cho biến `rand` từ 1 đến 4 để quyết định hướng di chuyển tiếp theo của kẻ thù.
- `self.max_travel = random.randint(30, 60)`: Chọn một giá trị ngẫu nhiên cho biến `max_travel`, đại diện cho số bước di chuyển tối đa của kẻ thù trước khi thay đổi hướng di chuyển. Điều này giúp tạo ra sự đa dạng trong hành vi di chuyển của kẻ thù.



#### 8. **taunted\_movement(self, distance):**

- Xử lý việc di chuyển của kẻ địch khi bị thách thức bởi người chơi.
- `player = self.game.player`: Lấy đối tượng người chơi từ trò chơi.
- `dy = player.rect.centery - self.rect.centery` và `dx = player.rect.centerx - self.rect.centerx`: Tính toán sự chênh lệch về vị trí theo chiều y và chiều x giữa kẻ thù và người chơi.
- `self.rad = math.atan2(dy, dx)`: Tính toán góc radian của vector từ kẻ thù tới người chơi sử dụng hàm `math.atan2()`.
- `if distance > self.weapon.scope`:: Kiểm tra xem khoảng cách giữa kẻ thù và người chơi có lớn hơn phạm vi bắn của vũ khí không.
- `self.x_change += ENEMY_SPEED * math.cos(self.rad)` và `self.y_change += ENEMY_SPEED * math.sin(self.rad)`: Nếu khoảng cách lớn hơn phạm vi bắn của vũ khí, kẻ thù sẽ di chuyển về phía người chơi với tốc độ `ENEMY_SPEED` theo hướng đã tính toán.
- `elif self.weapon.can_shoot()`: Nếu khoảng cách giữa kẻ thù và người chơi nhỏ hơn hoặc bằng phạm vi bắn của vũ khí và vũ khí có thể bắn, thì kẻ thù sẽ thực hiện hành động bắn.

#### 9. **kill(self):**

- Phương thức này được gọi khi kẻ địch bị loại bỏ khỏi trò chơi.
- Loại bỏ vũ khí của kẻ địch và kẻ địch khỏi tất cả các nhóm sprite trong trò chơi.

### 3.2.5 Xây dựng Class Boss:

```
class Boss(Enemy):
    def __init__(self, game, x, y, mapx, mapy, map):
        self.game = game
        Enemy.__init__(self, self.game, x, y, mapx, mapy, map, {"ak47": 1})

        self.width = BOSS_SIZE
        self.height = BOSS_SIZE
        self.image = self.game.boss_spritesheet.get_sprite(0, 2, self.width,
self.height)
        self.rect = self.image.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y

        self.right_animations = [self.game.boss_spritesheet.get_sprite(12,83,
self.width, self.height),
                                self.game.boss_spritesheet.get_sprite(93,75, self.width,
self.height),
                                self.game.boss_spritesheet.get_sprite(190, 75,
self.width, self.height)]
```

```

        self.left_animations = [self.game.boss_spritesheet.get_sprite(18, 160,
self.width, self.height),
                                self.game.boss_spritesheet.get_sprite(113, 160,
self.width, self.height),
                                self.game.boss_spritesheet.get_sprite(203, 160,
self.width, self.height)]
        self.max_hp = 20
        self.HP = self.max_hp

```

Lớp **Boss** là một lớp con của lớp **Enemy**, được sử dụng để đại diện cho một loại kẻ địch đặc biệt, có sức mạnh và máu lớn hơn so với kẻ địch thông thường. Dưới đây là giải thích về lớp **Boss**:

1. **\_\_init\_\_(self, game, x, y, mapx, mapy, map):**

- Phương thức khởi tạo của lớp **Boss**.
- Gọi phương thức khởi tạo của lớp cha **Enemy** để thiết lập các thuộc tính chung với kẻ địch.
- Thiết lập các thuộc tính riêng cho boss như hình ảnh, kích thước, máu, vũ khí và hình ảnh hoạt hình.
- Boss chỉ có một loại vũ khí là AK47, nên truyền vào một từ điển chỉ có một phần tử là "ak47" với trọng số là 1.

2. **right\_animations** và **left\_animations**:

- Là danh sách các hình ảnh được sử dụng để hoạt hình cho boss khi di chuyển sang phải và sang trái.
- Hình ảnh này được lấy từ sprite sheet của boss và được chọn một cách phù hợp để tạo hiệu ứng hoạt hình.

3. **max\_hp**:

- Được thiết lập là 20, đại diện cho máu tối đa của boss.
- Máu của boss được khởi tạo là máu tối đa khi boss được tạo ra.

### 3.2.6 Xây dựng Class Block:

```

class Block(pygame.sprite.Sprite):
    def __init__(self, game, x, y, mapx, mapy):

        self.game = game
        self._layer = BLOCK_LAYER
        self.groups = self.game.all_sprites, self.game.blocks
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.x = mapx + x * TILE_SIZE
        self.y = mapy + y * TILE_SIZE
        self.width = TILE_SIZE

```

```

        self.height = TILE_SIZE

        self.image = self.game.terrain_spritesheet.get_sprite(384, 576, self.width,
self.height)

        self.rect = self.image.get_rect()
        self.rect.x = self.x

        self.rect.y = self.y

```

Lớp Block đại diện cho các khối trong môi trường của trò chơi. Dưới đây là giải thích về lớp Block:

#### 1. `__init__(self, game, x, y, mapx, mapy):`

- Phương thức khởi tạo của lớp Block.
- Cài đặt các thuộc tính cơ bản như vị trí, kích thước và hình ảnh của khối.
- Đặt các nhóm mà khối này thuộc về, bao gồm cả nhóm tất cả các sprite và nhóm các khối.
- Hình ảnh của khối được lấy từ sprite sheet của môi trường và được chọn để hiển thị khối trong trò chơi.
- Thiết lập vị trí ban đầu của khối dựa trên tọa độ x và y được chuyển vào và tọa độ mapx và mapy.

### 3.2.7 Xây dựng Class Ground:

```

class Ground(pygame.sprite.Sprite):
    def __init__(self, game, x, y, mapx, mapy):
        self.game = game
        self._layer = GROUND_LAYER
        self.groups = self.game.all_sprites
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.x = mapx + x * TILE_SIZE
        self.y = mapy + y * TILE_SIZE
        self.width = TILE_SIZE
        self.height = TILE_SIZE

        self.image = self.game.terrain_spritesheet.get_sprite(64, 352, self.width,
self.height)

        self.rect = self.image.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y

```

Lớp Ground đại diện cho các ô trống trên bản đồ, đại diện cho không gian mà nhân vật có

thể di chuyển qua mà không gặp phải bất kỳ rào cản nào. Dưới đây là giải thích về lớp Ground:

#### 1. `__init__(self, game, x, y, mapx, mapy):`

- Phương thức khởi tạo của lớp Ground.
- Cài đặt các thuộc tính cơ bản như vị trí, kích thước và hình ảnh của mặt đất.
- Đặt nhóm mà mặt đất này thuộc về, bao gồm cả nhóm tất cả các sprite.
- Hình ảnh của mặt đất được lấy từ sprite sheet của môi trường và được chọn để hiển thị mặt đất trong trò chơi.
- Thiết lập vị trí ban đầu của mặt đất dựa trên tọa độ x và y được chuyển vào và tọa độ mapx và mapy.

### 3.2.8 Xây dựng Class Attack:

```
class Attack(pygame.sprite.Sprite):
    def __init__(self, game, x, y):
        self._layer = PLAYER_LAYER
        self.game = game
        self.x = x
        self.y = y
        self.width = WEAPON_SIZE
        self.height = WEAPON_SIZE
        self.groups = self.game.all_sprites, self.game.attacks
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.animation_loop = 0
        self.image = self.game.attack_spritesheet.get_sprite(0, 0, self.width,
self.height)

        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y

    def update(self):
        self.animate()
        self.collide()

    def collide(self):
        hits = pygame.sprite.spritecollide(self, self.game.enemies, True)

    def animate(self):
        direction = self.game.player.facing

        right_animations = [self.game.attack_spritesheet.get_sprite(0, 64,
self.width, self.height),
                             self.game.attack_spritesheet.get_sprite(32, 64,
self.width, self.height),
```

```

self.width, self.height), self.game.attack_spritesheet.get_sprite(64, 64,
self.width, self.height), self.game.attack_spritesheet.get_sprite(96, 64,
self.width, self.height), self.game.attack_spritesheet.get_sprite(128, 64,
self.width, self.height)]

    down_animations = [self.game.attack_spritesheet.get_sprite(0, 32,
self.width, self.height), self.game.attack_spritesheet.get_sprite(32, 32,
self.width, self.height), self.game.attack_spritesheet.get_sprite(64, 32,
self.width, self.height), self.game.attack_spritesheet.get_sprite(96, 32,
self.width, self.height), self.game.attack_spritesheet.get_sprite(128, 32,
self.width, self.height)]

    left_animations = [self.game.attack_spritesheet.get_sprite(0, 96,
self.width, self.height), self.game.attack_spritesheet.get_sprite(32, 96,
self.width, self.height), self.game.attack_spritesheet.get_sprite(64, 96,
self.width, self.height), self.game.attack_spritesheet.get_sprite(96, 96,
self.width, self.height), self.game.attack_spritesheet.get_sprite(128, 96,
self.width, self.height)]

    up_animations = [self.game.attack_spritesheet.get_sprite(0, 0, self.width,
self.height), self.game.attack_spritesheet.get_sprite(32, 0, self.width,
self.height), self.game.attack_spritesheet.get_sprite(64, 0, self.width,
self.height), self.game.attack_spritesheet.get_sprite(96, 0, self.width,
self.height), self.game.attack_spritesheet.get_sprite(128, 0, self.width,
self.height)]

    if direction == "up":
        self.image = up_animations[math.floor(self.animation_loop)]
    if direction == "down":
        self.image = down_animations[math.floor(self.animation_loop)]
    if direction == "left":
        self.image = left_animations[math.floor(self.animation_loop)]
    if direction == "right":
        self.image = right_animations[math.floor(self.animation_loop)]

    self.animation_loop += 0.5
    if self.animation_loop >= 5:
        self.kill()

```

#### 1. **\_\_init\_\_(self, game, x, y):**

- Phương thức khởi tạo của lớp Attack.
- Thiết lập các thuộc tính cơ bản như vị trí, kích thước và hình ảnh của tấn công.
- Đặt nhóm mà tấn công này thuộc về, bao gồm cả nhóm tất cả các sprite và nhóm các tấn công.
- Hình ảnh của tấn công được chọn từ sprite sheet của trò chơi và được cắt ra từ hình ảnh toàn bộ để tạo ra hình ảnh của tấn công.
- Đặt hình ảnh ban đầu và vị trí của tấn công.

#### 2. **update(self):**

- Phương thức cập nhật trạng thái của tấn công trong mỗi frame của trò chơi.
- Thực hiện việc hoạt hình và xử lý va chạm của tấn công.

#### 3. **collide(self):**

- Xử lý va chạm giữa tấn công và kẻ địch.
- Nếu tấn công va chạm với một kẻ địch, kẻ địch đó sẽ bị loại bỏ khỏi trò chơi.

#### 4. **animate(self):**

- Phương thức này điều khiển việc hoạt hình của tấn công dựa trên hướng mà nhân vật chính đang nhìn.
- Dựa vào hướng, chọn các frame hoạt hình tương ứng từ sprite sheet để hiển thị.
- Thực hiện việc chuyển frame bằng cách tăng giá trị của biến `animation_loop`. Nếu đã đến frame cuối cùng, tấn công sẽ bị loại bỏ khỏi trò chơi bằng cách gọi phương thức `kill()`.

### 3.2.9 Xây dựng Class Bullet:

#### 1. **\_\_init\_\_(self, game, heading, rad, dmg, rad\_offset, speed, owner):**

- Phương thức khởi tạo của lớp Bullet.
- Cài đặt các thuộc tính cơ bản như vị trí, kích thước, sát thương, góc bắn và tốc độ của đạn.
- Đặt nhóm mà đạn này thuộc về, dựa vào loại của owner. Nếu owner là một đối tượng Enemy, đạn sẽ thuộc nhóm `game.enemies_bullets`, ngược lại, đạn sẽ thuộc nhóm `game.bullets`.
- Tạo hình dạng cho đạn, được tạo thành từ một hình tròn màu vàng.
- Đặt vị trí ban đầu và hình dạng của đạn.

#### 2. **movement(self):**

- Phương thức này điều khiển chuyển động của đạn dựa trên góc bắn và tốc độ.
- `self.rect.x += self.speed * math.cos(self.rad)`: Di chuyển đạn theo phương ngang (chiều x) dựa trên tốc độ của nó và góc radian `self.rad`. Hàm `math.cos()` được sử dụng để tính toán thành phần ngang của vector di chuyển.
- `self.rect.y += self.speed * math.sin(self.rad)`: Di chuyển đạn theo phương dọc (chiều y) dựa trên tốc độ của nó và góc radian `self.rad`. Hàm `math.sin()` được sử dụng để tính toán thành phần dọc của vector di chuyển.
- `self.max_travel -= self.speed`: Giảm giá trị của `self.max_travel` đi `self.speed`, đại diện cho khoảng cách còn lại mà đạn có thể di chuyển trước khi đạt đến điểm cuối.
- `if self.max_travel <= 0`: Kiểm tra xem giá trị của `self.max_travel` có nhỏ hơn hoặc bằng 0 không, tức là đạn đã di chuyển đến điểm cuối của quỹ đạo di chuyển.
- `self.kill()`: Nếu `self.max_travel` nhỏ hơn hoặc bằng 0, đạn sẽ bị loại bỏ khỏi trò chơi bằng cách gọi phương thức `kill()`, điều này thường xảy ra khi đạn đã đi quá xa hoặc chạm vào một vật cản.

#### **update(self):**

- Phương thức cập nhật trạng thái của đạn trong mỗi frame của trò chơi.
- Thực hiện việc di chuyển và xử lý va chạm của đạn.

#### **3. collide\_blocks(self):**

- Xử lý va chạm giữa đạn với block và entrance (entrance là phần sảnh nối giữa 2 phòng).
- Nếu đạn va chạm với bất kỳ khối hoặc cửa nào, đạn sẽ bị loại bỏ khỏi trò chơi.

### **3.2.10 Xây dựng Class Gun:**

```
class Gun(pygame.sprite.Sprite):
    def update(self):
        self.rad = self.owner.rad
        self.animate()
        self.movement()

    def animate(self):
        next_image = self.shoot_animation[math.floor(self.animation_loop)].copy()
        if self.have_left_target():
            next_image = pygame.transform.flip(next_image.copy(), True, False)
            rad = self.rad + math.pi
            self.owner.facing = "left"
            if self.alive():
                self.game.all_sprites.change_layer(self, self.owner._layer - 1)
                self.game.guns.change_layer(self, self.owner._layer - 1)
            self.image = pygame.transform.rotate(next_image, math.degrees(-rad))
```

```

        else:
            self.owner.facing = "right"
            if self.alive():
                self.game.all_sprites.change_layer(self, self.owner._layer + 1)
                self.game.guns.change_layer(self, self.owner._layer + 1)
                self.image = pygame.transform.rotate(next_image,
math.degrees(-self.rad))
            self.image.set_colorkey(BLACK)
            self.rect = self.image.get_rect(center=self.rect.center)

            if self.owner.attacking:
                self.animation_loop += 0.5
                if self.animation_loop >= 5:
                    self.animation_loop = 0
                    self.owner.attacking = False

        def movement(self):
            if self.owner.facing == "right":
                self.rect.center = (self.owner.rect.centerx + self.place_right[0],
self.owner.rect.centery + self.place_right[1])
            if self.owner.facing == "left":
                self.rect.center = (self.owner.rect.centerx - self.place_right[0],
self.owner.rect.centery + self.place_right[1])

        def have_left_target(self):
            return (self.rad > -math.pi and self.rad < -math.pi/2) or (self.rad >=
math.pi/2 and self.rad <= math.pi)

        def shoot(self):
            Bullet(self.game, self.find_heading(), self.rad, self.bullet_dmg,
self.rad_offset, self.speed, self.owner)
            if isinstance(self.owner, Player):
                self.owner.mana -= self.manacost

        def can_shoot(self):
            now = pygame.time.get_ticks()
            if now - self.timer > self.delay and (isinstance(self.owner, Enemy) or
isinstance(self.owner, Enemy) or self.owner.mana >= self.manacost):
                self.timer = now
            return True
            return False

        def find_heading(self):
            center_image = (self.width/2, self.height/2)
            vector = (self.headpos[0] - center_image[0], self.headpos[1] -
center_image[1])
            hypotenuse = math.sqrt(vector[0]**2 + vector[1]**2)
            alpha = math.acos(vector[0]/hypotenuse)
            if self.owner.facing == "right":
                alpha = -alpha
            headx = self.rect.centerx + hypotenuse * math.cos(self.rad+alpha)
            heady = self.rect.centery + hypotenuse * math.sin(self.rad+alpha)
            return headx, heady

```



### 1. **update(self):**

- Phương thức cập nhật trạng thái của vũ khí trong mỗi frame của trò chơi.
- Cập nhật góc bắn và hoạt hình của vũ khí, cũng như việc di chuyển vũ khí theo nhân vật.

### 2. **animate(self):**

- Phương thức này điều khiển hoạt hình của vũ khí dựa trên hướng mà nhân vật đang nhìn.
- Chọn hình ảnh phù hợp từ sprite sheet để hiển thị vũ khí theo hướng.
- Cập nhật tầng của vũ khí để đảm bảo hiển thị đúng trên màn hình.

### 3. **movement(self):**

- Phương thức này điều khiển di chuyển của vũ khí, nó sẽ được đặt ở bên trái hoặc bên phải của nhân vật tùy thuộc vào hướng mà nhân vật nhìn.

### 4. **shoot(self):**

- Phương thức này được gọi khi vũ khí bắn.
- Tạo ra một đối tượng đạn mới và đặt nó vào trò chơi.

### 5. **can\_shoot(self):**

- Kiểm tra xem vũ khí có thể bắn được không.
- Xác định xem đã đến thời điểm bắn tiếp theo chưa dựa trên thời gian trôi qua và khả năng bắn của nhân vật (nếu là người chơi).

## 3.2.11 Xây dựng Class Glock, AK47, SNIPER:

```
class Glock(Gun):
    def __init__(self, game, owner, delay = PLAYER_GLOCK_DELAY, bullet_dmg =
PLAYER_GLOCK_DMG):
        self._layer = GUN_LAYER
        self.game = game
        self.owner = owner
        self.x = self.owner.rect.centerx
        self.y = self.owner.rect.centery
        self.width = 48
        self.height = 32
        self.groups = self.game.all_sprites, self.game.guns
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.animation_loop = 0
        self.image = self.game.glock_spritesheet.get_sprite(0, 0, self.width,
self.height)

        self.rect = self.image.get_rect()
        self.rect.center = (self.x, self.y)
```

```

        self.shoot_animation = [self.game.glock_spritesheet.get_sprite(0, 0,
self.width, self.height),
                                self.game.glock_spritesheet.get_sprite(48, 0,
self.width, self.height),
                                self.game.glock_spritesheet.get_sprite(96, 0,
self.width, self.height),
                                self.game.glock_spritesheet.get_sprite(144, 0,
self.width, self.height),
                                self.game.glock_spritesheet.get_sprite(192, 0,
self.width, self.height)]
        self.timer = 0
        self.rad = self.owner.rad
        self.scope = GLOCK_SCOPE
        self.delay = delay
        self.bullet_dmg = bullet_dmg
        self.rad_offset = 3

        #pos against player to place gun when facing right
        self.place_right = (8, 6)
        self.headpos = (40, 8)
        self.manacost = 0
        self.speed = GLOCK_BULLET_SPEED
        if isinstance(self.owner, Enemy) and isinstance(self.owner, Boss) == False:
            self.delay *= 4

class AK47(Gun):
    def __init__(self, game, owner, delay = PLAYER_AK47_DELAY, bullet_dmg =
PLAYER_AK47_DMG):
        self._layer = GUN_LAYER
        self.game = game
        self.owner = owner
        self.x = self.owner.rect.centerx
        self.y = self.owner.rect.centery
        self.width = 64
        self.height = 16
        self.groups = self.game.all_sprites, self.game.guns
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.animation_loop = 0
        self.image = self.game.ak47_spritesheet.get_sprite(0, 0, self.width,
self.height)

        self.rect = self.image.get_rect()
        self.rect.center = (self.x, self.y)

        self.shoot_animation = [self.game.ak47_spritesheet.get_sprite(0, 0,
self.width, self.height),
                                self.game.ak47_spritesheet.get_sprite(64, 0, self.width,
self.height),
                                self.game.ak47_spritesheet.get_sprite(128, 0,
self.width, self.height),
                                self.game.ak47_spritesheet.get_sprite(192, 0,
self.width, self.height),

```

```

        self.game.ak47_spritesheet.get_sprite(256, 0,
self.width, self.height)]
        self.timer = 0
        self.rad = self.owner.rad
        self.scope = AK47_SCOPE
        self.delay = delay
        self.bullet_dmg = bullet_dmg
        self.rad_offset = 4
        #pos against player to place gun when facing right
        self.place_right = (6, 4)
        self.headpos = (48, 4)
        self.manacost = 2
        self.speed = AK47_BULLET_SPEED
        if isinstance(self.owner, Enemy) and isinstance(self.owner, Boss) == False:
            self.delay *= 4

class Sniper(Gun):
    def __init__(self, game, owner, delay = PLAYER_SNIPER_DELAY, bullet_dmg =
PLAYER_SNIPER_DMG):
        self._layer = GUN_LAYER
        self.game = game
        self.owner = owner
        self.x = self.owner.rect.centerx
        self.y = self.owner.rect.centery
        self.width = 80
        self.height = 32
        self.groups = self.game.all_sprites, self.game.guns
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.animation_loop = 0
        self.image = self.game.sniper_spritesheet.get_sprite(0, 0, self.width,
self.height)

        self.rect = self.image.get_rect()
        self.rect.center = (self.x, self.y)

        self.shoot_animation = [self.game.sniper_spritesheet.get_sprite(0, 0,
self.width, self.height),
                                self.game.sniper_spritesheet.get_sprite(80, 0,
self.width, self.height),
                                self.game.sniper_spritesheet.get_sprite(160, 0,
self.width, self.height),
                                self.game.sniper_spritesheet.get_sprite(240, 0,
self.width, self.height),
                                self.game.sniper_spritesheet.get_sprite(320, 0,
self.width, self.height)]

        self.timer = 0
        self.rad = self.owner.rad
        self.scope = SNIPER_SCOPE
        self.delay = delay
        self.bullet_dmg = bullet_dmg
        self.rad_offset = 0

```

```
#pos against player to place gun when facing right
self.place_right = (8, 4)
self.headpos = (48, 6)
self.manacost = 5
self.speed = SNIPER_BULLET_SPEED
if isinstance(self.owner, Enemy) and isinstance(self.owner, Boss) == False:
    self.delay *= 4
```

- self.\_layer = GUN\_LAYER: Đặt layer của súng Glock trong trò chơi, cho phép xác định thứ tự vẽ các sprite trên màn hình.
- self.game = game: Lưu trữ tham chiếu đến đối tượng trò chơi.
- self.owner = owner: Lưu trữ tham chiếu đến chủ sở hữu của súng, tức là nhân vật hoặc kẻ thù sở hữu súng.
- self.x = self.owner.rect.centerx và self.y = self.owner.rect.centery: Xác định tọa độ x và y của súng dựa trên tọa độ trung tâm của chủ sở hữu.
- self.width = 48 và self.height = 32: Xác định kích thước của súng.
- self.groups = self.game.all\_sprites, self.game.guns: Xác định nhóm của sprite để quản lý việc vẽ và cập nhật.
- pygame.sprite.Sprite.\_\_init\_\_(self, self.groups): Gọi hàm khởi tạo của lớp cha Sprite để khởi tạo sprite.
- self.animation\_loop = 0: Biến này được sử dụng để theo dõi vòng lặp hoạt ảnh khi súng bắn.
- self.image = self.game.glock\_spritesheet.get\_sprite(0, 0, self.width, self.height): Tạo sprite của súng Glock sử dụng hình ảnh từ spritesheet.
- self.rect = self.image.get\_rect(): Tạo hình chữ nhật xung quanh sprite để xác định vị trí và kích thước của sprite.
- self.rect.center = (self.x, self.y): Đặt tọa độ trung tâm của hình chữ nhật sprite để là tọa độ của chủ sở hữu.
- self.shoot\_animation = [...]: Định nghĩa danh sách các hình ảnh để tạo hiệu ứng bắn của súng Glock.
- self.timer = 0: Biến thời gian để đếm thời gian giữa các lần bắn.
- self.rad = self.owner.rad: Góc radian của súng được đặt là góc radian của chủ sở hữu.
- self.scope = GLOCK\_SCOPE: Phạm vi của súng Glock.
- self.delay = delay: Thời gian chờ giữa các lần bắn, mặc định là thời gian chờ của người chơi.
- self.bullet\_dmg = bullet\_dmg: Lượng sát thương của đạn.
- self.rad\_offset = 3: Độ lệch góc radian cho phép đạn bắn ra so với góc radian của chủ sở hữu.
- Tương tự với AK47 và SNIPER

### 3.2.12 Xây dựng Class Button:

```
class Button:
    def __init__(self, x, y, width, height, fg, bg, content, fontsize):
        self.font = pygame.font.Font('pygameRPG/Arial.ttf', fontsize)
        self.content = content
        self.x = x
        self.y = y
        self.width = width
        self.height = height
```

```

        self.width = width
        self.fg = fg

        self.image = pygame.Surface((self.width, self.height))
        self.image.fill(bg)
        self.rect = self.image.get_rect()

        self.rect.x = self.x
        self.rect.y = self.y
        self.text = self.font.render(self.content, True, self.fg)
        self.text_rect = self.text.get_rect(center=(self.width/2, self.height/2))
        self.image.blit(self.text, self.text_rect)

    def isPressed(self, mousepos, pressed):
        if self.rect.collidepoint(mousepos):
            if pressed[0]:
                return True
        return False

```

- **\_\_init\_\_(...):**
  - Phương thức khởi tạo nhận các đối số như tọa độ, kích thước, màu sắc, nội dung, và cỡ chữ của nút.
  - Nút được tạo dưới dạng một hình chữ nhật với màu nền và kích thước được chỉ định.
  - Nội dung của nút được hiển thị bằng cách sử dụng font và màu chữ được chỉ định.
  - Hình ảnh và vị trí của nút được cập nhật dựa trên thông số đã được cung cấp.
- **isPressed(...)**
  - Phương thức này kiểm tra xem nút có được nhấn hay không dựa trên vị trí của chuột và trạng thái nút chuột.
  - Nếu chuột được nhấn và nó đang ở trên nút, phương thức trả về True, ngược lại trả về False.

### 3.2.13 Xây dựng Class MyMap:

```

class MyMap(pygame.sprite.Sprite):
    def __init__(self, tilemap, game, phase_num = 0):
        self._layer = MAP_LAYER
        self.mappingpos = [0, 0]
        self.tilemap = tilemap
        self.isDrawn = False
        self.game = game
        #start position of the map including the border
        self.x = 0
        self.y = 0

```

```

        #display a invisible rect to check if player is in the map, it not include
the border(32 pixels block)
        self.image = pygame.Surface((WIN_WIDTH - TILE_SIZE*2, WIN_HEIGHT -
TILE_SIZE*2))
        self.rect = self.image.get_rect()

        self.top = None
        self.bottom = None
        self.left = None
        self.right = None

        self.num_enemies = 0
        self.groups = self.game.all_sprites
        pygame.sprite.Sprite.__init__(self, self.groups)
        self.entrances = pygame.sprite.Group()
        self.enemies = pygame.sprite.Group()
        self.open = True
        self.max_phase = phase_num
        self.current_phase = 0
        self.available_pos = []
        self.clear = False
        self.current_phase = 1

    def draw(self):
        self.create_tilemap()
        self.isDrawn = True

    def checkEntrance(self, i, j):
        if(self.top != None):
            if i == 0 and j > 6 and j < 13:
                return True
        if(self.bottom != None):
            if i == 14 and j > 6 and j < 13:
                return True
        if(self.left != None):
            if j == 0 and i > 4 and i < 10:
                return True
        if(self.right != None):
            if j == 19 and i > 4 and i < 10:
                return True
        return False

    def check_corner(self, i, j):
        if (i, j) in [(0,0), (0,19), (14,0), (14,19)]:
            return True
        if(self.top != None):
            if (i,j) in [(0,6), (0,13)]:
                return True
        if(self.bottom != None):
            if (i,j) in [(14,6), (14,13)]:
                return True
        if(self.left != None):
            if (i,j) in [(4, 0), (10,0)]:

```

```

        return True
    if(self.right != None):
        if (i,j) in [(4,19), (10,19)]:
            return True
    return False

def update(self):
    self.display_hp_boss()
    self.update_phase()
    if self.rect.contains(self.game.player.rect) and self.enemies:
        for entrance in self.entrances:
            entrance.enable = True
        self.open = False
    else:
        for entrance in self.entrances:
            entrance.enable = False
        self.open = True

def update_phase(self):
    #get num enemy in self.enemies
    num_enemies = len(self.enemies.sprites())

    if(num_enemies == 0 and self.current_phase >= self.max_phase):
        self.clear = True
    if( num_enemies == 0 and self.current_phase < self.max_phase):
        self.current_phase += 1
        #chon 5 vi tri random de tao enemy
        enemies_pos = random.sample(self.available_pos, random.randint(2, 8))
        for pos in enemies_pos:
            enemy = Enemy(self.game, pos[0], pos[1], self.rect.x-32,
self.rect.y-32, self)
            self.enemies.add(enemy)

def update_rect(self):
    #start position of the map including the border
    self.x = self.mappingpos[0] * WIN_WIDTH
    self.y = self.mappingpos[1] * WIN_HEIGHT
    self.rect.topleft = ( 32 + self.mappingpos[0] * WIN_WIDTH, 32 +
self.mappingpos[1] * WIN_HEIGHT)

def create_tilemap(self):
    for i, row in enumerate(self.tilemap):
        for j, col in enumerate(row):
            if col == 'B':
                if(self.checkEntrance(i, j)):
                    sprite = Entrance(self.game, j, i, self.x, self.y)
                    self.entrances.add(sprite)
                else: Block(self.game, j, i, self.x, self.y)
                if self.check_corner(i, j):
                    Torch(self.game, j, i, self.x, self.y)
            if col == 'E':
                enemy = Enemy(self.game, j, i, self.x, self.y, self)
                self.enemies.add(enemy)

```

```

        if col == 'P':
            self.game.player = Player(self.game, j, i, self.x, self.y)
            self.game.player.set_weapons()
        if col == 'E' or col == 'P' or col == '.':
            self.available_pos.append([j, i])
        if col == '5':
            boss = Boss(self.game, j, i, self.x, self.y, self)
            self.enemies.add(boss)
        if(col == ' '): continue
        Ground(self.game, j, i, self.x, self.y)

def get_boss(self):
    for enemy in self.enemies:
        if isinstance(enemy, Boss):
            return enemy
    return None

def display_hp_boss(self):
    if self.rect.contains(self.game.player.rect) and self.enemies:
        boss = self.get_boss()
        if boss:
            BossHPBar(self.game, boss )

```

- **\_\_init\_\_(...):**
  - Phương thức khởi tạo nhận các đối số như tilemap, game (trò chơi), và phase\_num (số giai đoạn).
  - Cài đặt các biến và thuộc tính, bao gồm các nhóm sprite cho các cửa, kẻ thù và tất cả các sprite.
  - Khởi tạo một phần của bản đồ dưới dạng một hình chữ nhật vô hình để kiểm tra xem người chơi có ở trong bản đồ không.
  - Xác định các cửa và kẻ thù trên bản đồ dựa trên dữ liệu tilemap.
- **draw():**
  - Tạo phần của bản đồ bằng cách gọi phương thức create\_tilemap().
- **update():**
  - Cập nhật trạng thái của bản đồ dựa trên vị trí của người chơi và số lượng kẻ thù còn lại.
  - Kích hoạt hoặc vô hiệu hóa các cửa tùy thuộc vào vị trí của người chơi.
- **update\_phase():**
  - Cập nhật giai đoạn hiện tại của trò chơi dựa trên số lượng kẻ thù còn lại.
  - Tạo ra các kẻ thù mới nếu tất cả đã bị tiêu diệt.
- **create\_tilemap():**
  - Tạo ra các sprite cho phần bản đồ dựa trên dữ liệu tilemap.



- Xác định vị trí của người chơi, cửa, kẻ thù và các vị trí trống trên bản đồ.
- **get\_boss():**
  - Trả về kẻ thù loại Boss nếu có trong danh sách kẻ thù.
- **display\_hp\_boss():**
  - Hiển thị thanh máu của Boss nếu người chơi đang ở trong bản đồ và Boss còn sống.
- **checkEntrance():**
- if(self.top != None):: Kiểm tra xem ô hiện tại có lối vào phía trên không (tức là có cửa ở phía trên).
- if i == 0 and j > 6 and j < 13:: Nếu ô hiện tại là hàng đầu tiên (i == 0), và cột j của ô đang xét nằm trong khoảng từ cột thứ 7 đến cột thứ 12, thì được xem là một lối vào hợp lệ.
- if(self.bottom != None):: Kiểm tra xem ô hiện tại có lối vào phía dưới không (tức là có cửa ở phía dưới).
- if i == 14 and j > 6 and j < 13:: Nếu ô hiện tại là hàng cuối cùng (i == 14), và cột j của ô đang xét nằm trong khoảng từ cột thứ 7 đến cột thứ 12, thì được xem là một lối vào hợp lệ.
- if(self.left != None):: Kiểm tra xem ô hiện tại có lối vào phía trái không (tức là có cửa ở phía trái).
- if j == 0 and i > 4 and i < 10:: Nếu ô hiện tại là cột đầu tiên (j == 0), và hàng i của ô đang xét nằm trong khoảng từ hàng thứ 5 đến hàng thứ 9, thì được xem là một lối vào hợp lệ.
- if(self.right != None):: Kiểm tra xem ô hiện tại có lối vào phía phải không (tức là có cửa ở phía phải).
- if j == 19 and i > 4 and i < 10:: Nếu ô hiện tại là cột cuối cùng (j == 19), và hàng i của ô đang xét nằm trong khoảng từ hàng thứ 5 đến hàng thứ 9, thì được xem là một lối vào hợp lệ.
- return False: Nếu không có lối vào nào được tìm thấy, hàm sẽ trả về False.

### 3.2.14 Xây dựng Class MapList:

```
class MapList:
    def __init__(self, tilemaps, game):
        self.maps = []
        self.pipes = []
        self.game = game
        for tilemap in tilemaps:
            self.maps.append(MyMap(tilemap, game, 2))
```

```

self.maps[0].max_phase = 0
self.maps[1].max_phase = 0
self.link(self.maps[0], self.maps[1], "right")
self.link(self.maps[1], self.maps[2], "top")
self.link(self.maps[1], self.maps[3], "bottom")
self.link(self.maps[1], self.maps[4], "right")
self.link(self.maps[4], self.maps[5], "right")

def draw(self):
    self.DFS_draw(self.maps[0])

def DFS_draw(self, map):
    map.draw()
    for adj in [map.top, map.bottom, map.left, map.right]:
        if adj != None and not adj.isDrawn:
            self.DFS_draw(adj)

def link(self, m1, m2, dir):
    pipe = MyMap(hpipemap, self.game)
    self.pipes.append(pipe)

    if(dir == "right"):
        pipe.tilemap = hpipemap
        m1.right = pipe
        pipe.right = m2
        m2.left = pipe
        m2.mappingpos = [m1.mappingpos[0] + 2, m1.mappingpos[1]]
        pipe.mappingpos = [m1.mappingpos[0] + 1, m1.mappingpos[1]]
    if(dir == "left"):
        pipe.tilemap = hpipemap
        m1.left = pipe
        pipe.left = m2
        m2.right = pipe
        m2.mappingpos = [m1.mappingpos[0] - 2, m1.mappingpos[1]]
        pipe.mappingpos = [m1.mappingpos[0] - 1, m1.mappingpos[1]]
    if(dir == "top"):
        pipe.tilemap = vpipemap
        m1.top = pipe
        pipe.top = m2
        m2.bottom = pipe
        m2.mappingpos = [m1.mappingpos[0], m1.mappingpos[1] - 2]
        pipe.mappingpos = [m1.mappingpos[0], m1.mappingpos[1] - 1]
    if(dir == "bottom"):
        pipe.tilemap = vpipemap
        m1.bottom = pipe
        pipe.bottom = m2
        m2.top = pipe
        m2.mappingpos = [m1.mappingpos[0], m1.mappingpos[1] + 2]
        pipe.mappingpos = [m1.mappingpos[0], m1.mappingpos[1] + 1]

    m1.update_rect()
    m2.update_rect()

```

```
pipe.update_rect()
```

```
def check_win(self):
    #neu boss chet thi win
    for map in self.maps:
        if map.get_boss() != None:
            return False
    return True
```

- **\_\_init\_\_(...):**
  - Phương thức khởi tạo nhận danh sách các tilemap và trò chơi (game) làm đối số.
  - Tạo một danh sách (self.maps) chứa các bản đồ (MyMap) dựa trên danh sách các tilemap đã cho.
  - Thiết lập các giá trị max\_phase cho các bản đồ.
  - Liên kết các bản đồ với nhau bằng cách sử dụng phương thức link().
- **draw():**
  - Duyệt qua tất cả các bản đồ và vẽ chúng bằng cách gọi phương thức DFS\_draw().
- **DFS\_draw(self, map):**
  - Duyệt đệ quy qua tất cả các bản đồ kề nhau và vẽ chúng nếu chưa được vẽ.
- **link(self, m1, m2, dir):**
  - Liên kết hai bản đồ m1 và m2 với nhau theo hướng dir ("left", "right", "top", hoặc "bottom").
  - Tạo một ống (pipe) để liên kết giữa hai bản đồ.
  - Cập nhật vị trí và kích thước của các bản đồ và ống.
- **check\_win(self):**
  - Kiểm tra xem nếu không còn Boss trên bản đồ thì trả về true, ngược lại trả về false.

### 3.2.15 Xây dựng Class PlayerBar:

```
class PlayerBars(pygame.sprite.Sprite):
    def __init__(self, game):
        self._layer = BAR_LAYER
        self.groups = game.bars
        pygame.sprite.Sprite.__init__(self, self.groups)
        self.game = game
        self.font = pygame.font.Font('pygameRPG/Arial.ttf', 16)
        self.icon_size = (20, 18)
        self.bar_size = (100, 18)
```

```

        self.gap = 8

        self.image = pygame.Surface((self.icon_size[0] + self.bar_size[0] +
self.gap*3, self.icon_size[1]*3 + self.gap*4))
        self.rect = self.image.get_rect()
        self.rect.x = 10
        self.rect.y = 10

        self.health_icon = pygame.Surface(self.icon_size)
        self.health_icon.fill(RED)
        self.armor_icon = pygame.Surface(self.icon_size)
        self.armor_icon.fill(GREY)
        self.mana_icon = pygame.Surface(self.icon_size)
        self.mana_icon.fill(BLUE)

        self.health_icon_rect = self.health_icon.get_rect()
        self.health_icon_rect.topleft = (self.gap, self.gap)
        self.armor_icon_rect = self.armor_icon.get_rect()
        self.armor_icon_rect.topleft = (self.health_icon_rect.left,
self.health_icon_rect.bottom + self.gap)
        self.mana_icon_rect = self.mana_icon.get_rect()
        self.mana_icon_rect.topleft = (self.health_icon_rect.left,
self.armor_icon_rect.bottom + self.gap)

        self.health_bg = pygame.Surface(self.bar_size)
        self.health_bg.fill(DARK_BROWN)
        self.armor_bg = pygame.Surface(self.bar_size)
        self.armor_bg.fill(DARK_BROWN)
        self.mana_bg = pygame.Surface(self.bar_size)
        self.mana_bg.fill(DARK_BROWN)

        self.health_bg_rect = self.health_bg.get_rect()
        self.armor_bg_rect = self.armor_bg.get_rect()
        self.mana_bg_rect = self.mana_bg.get_rect()
        self.health_bg_rect.topleft = (self.health_icon_rect.right + self.gap,
self.health_icon_rect.top)
        self.armor_bg_rect.topleft = (self.armor_icon_rect.right + self.gap,
self.armor_icon_rect.top)
        self.mana_bg_rect.topleft = (self.mana_icon_rect.right + self.gap,
self.mana_icon_rect.top)

        self.health_bar = pygame.Surface(self.bar_size)
        self.health_bar.fill(RED)
        self.armor_bar = pygame.Surface(self.bar_size)
        self.armor_bar.fill(GREY)
        self.mana_bar = pygame.Surface(self.bar_size)
        self.mana_bar.fill(BLUE)

        #rect doi voi cac bg tuong ung
        self.health_bar_rect = self.health_bar.get_rect()
        self.health_bar_rect.topleft = (0 ,0)
        self.armor_bar_rect = self.armor_bar.get_rect()
        self.armor_bar_rect.topleft = (0 ,0)

```

```

        self mana_bar_rect = self mana_bar.get_rect()
        self mana_bar_rect.topleft = (0,0)

        self.health_bg.blit(self.health_bar, self.health_bar_rect)
        self. armour_bg.blit(self. armour_bar, self. armour_bar_rect)
        self. mana_bg.blit(self. mana_bar, self. mana_bar_rect)

        self.image.fill(BROWN)
        self.image.blit(self.health_bg, self.health_bg_rect)
        self.image.blit(self. armour_bg, self. armour_bg_rect)
        self.image.blit(self. mana_bg, self. mana_bg_rect)
        self.image.blit(self.health_icon, self.health_icon_rect)
        self.image.blit(self. armour_icon, self. armour_icon_rect)
        self.image.blit(self. mana_icon, self. mana_icon_rect)

    def update(self):
        self.draw_HP()
        self.draw_AR()
        self.draw_MP()
        pass

    def draw_HP(self):
        self.health_bg.fill(DARK_BROWN)
        self.health_bar_rect.right = self.bar_size[0] * self.game.player.HP /
self.game.player.max_hp
        info = self.font.render(f"{self.game.player.HP}/{self.game.player.max_hp}",
True, WHITE)
        info_rect = info.get_rect()
        info_rect.center = self.health_bg_rect.center
        self.health_bg.blit(self.health_bar, self.health_bar_rect)
        self.image.blit(self.health_bg, self.health_bg_rect)
        self.image.blit(info, info_rect)

    def draw_AR(self):
        self. armour_bg.fill(DARK_BROWN)
        self. armour_bar_rect.right = self.bar_size[0] * self.game.player. armour /
self.game.player.max_ armour
        info =
self.font.render(f"{self.game.player. armour}/{self.game.player.max_ armour}", True,
WHITE)
        info_rect = info.get_rect()
        info_rect.center = self. armour_bg_rect.center
        self. armour_bg.blit(self. armour_bar, self. armour_bar_rect)
        self.image.blit(self. armour_bg, self. armour_bg_rect)
        self.image.blit(info, info_rect)

    def draw_MP(self):
        self. mana_bg.fill(DARK_BROWN)
        self. mana_bar_rect.right = self.bar_size[0] * self.game.player.mana /
self.game.player.max_mana
        info =
self.font.render(f"{self.game.player.mana}/{self.game.player.max_mana}", True,
WHITE)

```

```

info_rect = info.get_rect()
info_rect.center = self.mana_bg_rect.center
self.mana_bg.blit(self.mana_bar, self.mana_bar_rect)
self.image.blit(self.mana_bg, self.mana_bg_rect)
self.image.blit(info, info_rect)

```

**Lớp PlayerBars quản lý thanh trạng thái của người chơi trong trò chơi, bao gồm thanh máu, giáp và mana. Dưới đây là mô tả của các phương thức trong lớp này:**

- **\_\_init\_\_(...):**
  - Phương thức khởi tạo nhận trò chơi (game) làm đối số.
  - Tạo hình ảnh cho thanh trạng thái của người chơi bao gồm thanh máu, giáp và mana.
- **update():**
  - Cập nhật hiển thị cho thanh trạng thái của người chơi.
- **draw\_HP(), draw\_AR(), draw\_MP():**
  - Cập nhật hiển thị cho thanh máu, giáp và mana của người chơi.

### 3.2.16 Xây dựng Class BossHPBar:

```

class BossHPBar(pygame.sprite.Sprite):
    def __init__(self, game, owner):
        self._layer = BAR_LAYER
        self.groups = game.bars
        pygame.sprite.Sprite.__init__(self, self.groups)
        self.game = game
        self.bar_size = (240, 20)
        self.border = 2
        self.owner = owner

        self.image = pygame.Surface((self.bar_size[0]+self.border*2,
self.bar_size[1]+self.border*2))
        self.rect = self.image.get_rect()
        self.rect.centerx = WIN_WIDTH//2
        self.rect.y = 60

        self.health_bg = pygame.Surface(self.bar_size)
        self.health_bg.fill((71, 62, 62))
        self.health_bar = pygame.Surface(self.bar_size)
        self.health_bar.fill(RED)
        self.health_bar_rect = self.health_bar.get_rect()
        self.health_bar_rect.topleft = (0,0)
        self.health_bg_rect = self.health_bg.get_rect()
        self.health_bg_rect.topleft = (self.border, self.border)

        self.health_bg.blit(self.health_bar, self.health_bar_rect)

```

```

        self.image.fill(BLACK)
        self.image.blit(self.health_bg, self.health_bg_rect)

    def update(self):
        self.draw_HP()
        pass

    def draw_HP(self):
        self.health_bar_rect.right = self.bar_size[0] * self.owner.HP /
self.owner.max_hp

        self.health_bg.fill((43, 40, 40))
        self.health_bg.blit(self.health_bar, self.health_bar_rect)
        self.image.blit(self.health_bg, self.health_bg_rect)
        if self.owner.HP <= 0:
            self.kill()

```

- **\_\_init\_\_(self, game, owner):**
  - Phương thức khởi tạo nhận trò chơi (game) và đối tượng chủ sở hữu (Boss) của thanh hiển thị làm đối số.
  - Tạo hình ảnh cho thanh hiển thị sức khỏe của Boss bao gồm thanh nền và thanh sức khỏe.
  - Thiết lập kích thước và vị trí cho thanh hiển thị.
- **update(self):**
  - Cập nhật hiển thị cho thanh hiển thị sức khỏe của Boss.
- **draw\_HP(self):**
  - Cập nhật thanh sức khỏe của Boss dựa trên sức khỏe hiện tại và sức khỏe tối đa của Boss.
  - Vẽ lại thanh nền và thanh sức khỏe mới với sức khỏe hiện tại của Boss.
  - Nếu sức khỏe của Boss giảm xuống dưới 0, thì thanh hiển thị sẽ bị loại bỏ (kill).

### 3.2.17 Xây dựng Class Torch:

```

class Torch(pygame.sprite.Sprite):
    def __init__(self, game, x, y, mapx, mapy):
        self.game = game
        self._layer = TORCH_LAYER
        self.groups = self.game.all_sprites
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.x = mapx + x * TILE_SIZE
        self.y = mapy + y * TILE_SIZE
        self.width = 32
        self.height = 32

        self.animations = [pygame.image.load("pygameRPG/img/torch_1.png").convert(),

```

```

        pygame.image.load("pygameRPG/img/torch_2.png").convert(),
        pygame.image.load("pygameRPG/img/torch_3.png").convert(),
        pygame.image.load("pygameRPG/img/torch_4.png").convert()
    ]

    for i in range(4):
        image = pygame.transform.scale(self.animations[i], (64,64))
        image.set_colorkey(BLACK)
        self.animations[i] = image
    self.animation_loop = 0

    self.image = self.animations[0]
    self.rect = self.image.get_rect()
    self.rect.x = self.x - 16
    self.rect.y = self.y - 32

    self.timer = pygame.time.get_ticks()

def update(self):
    if pygame.time.get_ticks() - self.timer > 100:
        if self.animation_loop > 3:
            self.animation_loop = 0
        self.image = self.animations[self.animation_loop]
        self.animation_loop += 1
        self.timer = pygame.time.get_ticks()

```

Lớp Torch được sử dụng để tạo ra hiệu ứng đèn lò trong trò chơi. Dưới đây là mô tả của các phương thức trong lớp này:

- **\_\_init\_\_(self, game, x, y, mapx, mapy):**
  - Phương thức khởi tạo nhận các tham số là trò chơi (game), tọa độ x và y của torch trên màn hình (x, y), và tọa độ của torch trên bản đồ (mapx, mapy).
  - Thiết lập các biến cần thiết như game, layer, groups và các animation cho torch.
  - Thiết lập hình ảnh và vị trí cho torch.
- **update(self):**
  - Cập nhật hình ảnh của torch để tạo hiệu ứng chớp sáng.
  - Hiệu ứng này sẽ thay đổi hình ảnh của torch sau mỗi khoảng thời gian nhất định, tạo ra cảm giác lò sáng lung linh.

### 3.3 Xây dựng File khởi tạo trò chơi:

Import:

```

import pygame
from sprites import *
from config import *

```



```
import sys
```

class Game:

```
def __init__(self):
    pygame.init()
    self.screen = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))
    self.clock = pygame.time.Clock()
    self.font = pygame.font.Font('pygameRPG/Arial.ttf', 32)
    self.running = True

    self.character_spritesheet = Spritesheet('pygameRPG/img/character.png')
    self.terrain_spritesheet = Spritesheet('pygameRPG/img/terrain.png')
    self.enemy_spritesheet = Spritesheet('pygameRPG/img/enemy.png')
    self.attack_spritesheet = Spritesheet('pygameRPG/img/attack.png')
    self.intro_background =
pygame.image.load('pygameRPG/img/introbackground.png')
    self.gameover_background = pygame.image.load('pygameRPG/img/gameover.png')
    self.glock_spritesheet = Spritesheet('pygameRPG/img/Glock-SpriteSheet.png')
    self.ak47_spritesheet = Spritesheet('pygameRPG/img/AK47-SpriteSheet.png')
    self.sniper_spritesheet =
Spritesheet('pygameRPG/img/SniperRifle-SpriteSheet.png')
    self.boss_spritesheet = Spritesheet('pygameRPG/img/boss1.jpg')

    self.t1 = pygame.time.get_ticks()
    self.player: Player = None
```

**pygame.init():**

- Khởi tạo thư viện pygame, chuẩn bị cho việc sử dụng các thành phần của pygame.
- **self.screen = pygame.display.set\_mode((WIN\_WIDTH, WIN\_HEIGHT)):**
  - Tạo cửa sổ màn hình với kích thước được xác định bởi **WIN\_WIDTH** và **WIN\_HEIGHT**.
  - **WIN\_WIDTH** và **WIN\_HEIGHT** được giả định là chiều rộng và chiều cao mong muốn của cửa sổ trò chơi.
- **self.clock = pygame.time.Clock():**
  - Tạo đối tượng đồng hồ để giữ cho trò chơi chạy ở tốc độ ổn định trên mọi hệ thống.
- **self.font = pygame.font.Font('pygameRPG/Arial.ttf', 32):**
  - Tạo đối tượng font để sử dụng trong trò chơi. Font được sử dụng ở đây là Arial với kích thước 32.
- **self.running = True:**
  - Biến **running** được sử dụng để kiểm soát việc chạy của trò chơi. Khi **running** là True, trò chơi vẫn đang chạy; khi **running** là False, trò chơi sẽ kết thúc.

- **Các dòng gán đối tượng spritesheet và background:**
  - Các dòng mã này tạo các đối tượng spritesheet và background, được sử dụng để tải hình ảnh cho các đối tượng trong trò chơi như nhân vật, môi trường, quái vật, vũ khí và các hình ảnh nền.
  - **Spritesheet** là một đối tượng được tạo ra từ một tập tin hình ảnh, được sử dụng để cắt và lấy các phần của hình ảnh cụ thể để sử dụng trong trò chơi.
- **self.t1 = pygame.time.get\_ticks():**
  - Lưu thời điểm bắt đầu của trò chơi (được tính bằng milliseconds) để sử dụng cho việc đo thời gian trong trò chơi.
- **self.player: Player = None:**
  - Khởi tạo biến **player** và gán giá trị ban đầu là **None**.
  - Biến này là một tham chiếu đến đối tượng người chơi trong trò chơi, được sử dụng để theo dõi và điều khiển người chơi trong các phần khác nhau của mã.

```

• def create_tilemap(self):
•     self.maps = MapList(tilemaps, self)
•     self.maps.draw()

```

Phương thức **create\_tilemap** trong class **Game** khởi tạo một đối tượng **MapList** để quản lý danh sách các bản đồ trong trò chơi và vẽ chúng trên màn hình. Dưới đây là giải thích về phương thức này:

- **self.maps = MapList(tilemaps, self):**
  - Tạo một đối tượng **MapList**, một danh sách các bản đồ trong trò chơi, với danh sách **tilemaps** được chuyển vào và tham chiếu đến đối tượng **Game** hiện tại (**self**).
  - Biến **tilemaps** được giả định là một danh sách các tilemap (bản đồ) được sử dụng trong trò chơi.
- **self.maps.draw():**
  - Gọi phương thức **draw()** của đối tượng **MapList**.
  - Phương thức này sẽ kích hoạt việc vẽ các bản đồ trong danh sách ra màn hình.

```

# new game start
def new(self):
    self.playing = True
    self.all_sprites = pygame.sprite.LayeredUpdates()
    self.blocks = pygame.sprite.LayeredUpdates()
    self.enemies = pygame.sprite.LayeredUpdates()
    self.attacks = pygame.sprite.LayeredUpdates()
    self.bullets = pygame.sprite.LayeredUpdates()
    self.guns = pygame.sprite.LayeredUpdates()

```

```

self.entrances = pygame.sprite.LayeredUpdates()
self.enemies_bullets = pygame.sprite.LayeredUpdates()

#player health and armor bar
self.bars = pygame.sprite.LayeredUpdates()
self.create_tilemap()
PlayerBars(self)

```

- **self.playing = True:**
  - Đặt cờ **playing** thành **True**, chỉ ra rằng trò chơi đang chạy.
- **Khởi tạo các nhóm sprite:**
  - **self.all\_sprites:** Nhóm chứa tất cả các sprite trong trò chơi.
  - **self.blocks:** Nhóm chứa các sprite đại diện cho các khối vật lý trong trò chơi.
  - **self.enemies:** Nhóm chứa các sprite đại diện cho các kẻ thù trong trò chơi.
  - **self.attacks:** Nhóm chứa các sprite đại diện cho các tấn công của nhân vật trong trò chơi.
  - **self.bullets:** Nhóm chứa các sprite đại diện cho đạn của nhân vật trong trò chơi.
  - **self.guns:** Nhóm chứa các sprite đại diện cho các vũ khí trong trò chơi.
  - **self.entrances:** Nhóm chứa các sprite đại diện cho cửa vào trong các bản đồ của trò chơi.
  - **self.enemies\_bullets:** Nhóm chứa các sprite đại diện cho đạn của kẻ thù trong trò chơi.
- **Khởi tạo thanh máu và giáp của người chơi:**
  - **self.bars:** Nhóm chứa các thanh máu và giáp của người chơi.
  - Gọi **create\_tilemap()** để tạo bản đồ cho trò chơi mới.
  - Tạo một đối tượng **PlayerBars** để hiển thị thanh máu và giáp của người chơi trên màn hình.

```

def events(self):
    #game loop events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.playing = False
            self.running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                if self.player.facing == 'right':
                    Attack(self, self.player.rect.x + 32, self.player.rect.y)
                if self.player.facing == 'left':
                    Attack(self, self.player.rect.x - 32, self.player.rect.y)
                if self.player.facing == 'up':

```

```

        Attack(self, self.player.rect.x, self.player.rect.y - 32)
        if self.player.facing == 'down':
            Attack(self, self.player.rect.x, self.player.rect.y + 32)
    if event.key == pygame.K_1:
        self.player.change_weapon(0)
    elif event.key == pygame.K_2:
        self.player.change_weapon(1)
    elif event.key == pygame.K_3:
        self.player.change_weapon(2)
    if event.type == pygame.MOUSEBUTTONDOWN:
        if event.button == 1 and self.player.weapon != None and
self.player.weapon.can_shoot():
            self.player.attacking = True
            self.player.weapon.shoot()
        pass

```

Phương thức **events** trong class **Game** được sử dụng để xử lý các sự kiện trong trò chơi. Dưới đây là giải thích về phương thức này:

- **Xử lý sự kiện thoát trò chơi:**
  - Nếu sự kiện là **pygame.QUIT**, tức là người chơi bấm nút đóng cửa sổ, thì **self.playing** và **self.running** sẽ được đặt thành **False**, dừng trò chơi.
- **Xử lý sự kiện nhấn phím:**
  - Nếu người chơi nhấn phím cách (**pygame.K\_SPACE**), một đối tượng tấn công mới (**Attack**) sẽ được tạo và đặt ở vị trí tương ứng với hướng mà người chơi đang nhìn.
  - Nếu người chơi nhấn các phím số (**pygame.K\_1**, **pygame.K\_2**, **pygame.K\_3**), phương thức **change\_weapon** của người chơi sẽ được gọi để thay đổi vũ khí.
- **Xử lý sự kiện nhấn chuột:**
  - Nếu người chơi nhấn nút trái của chuột và vũ khí của người chơi có thể bắn (**can\_shoot()**), người chơi sẽ bắn ra một viên đạn từ vũ khí hiện tại.

Các sự kiện này giúp điều khiển hành vi của người chơi trong trò chơi.

```

def update(self):
    self.all_sprites.update()
    self.bars.update()

```

Phương thức **update** trong class **Game** cập nhật trạng thái của tất cả các đối tượng trong trò chơi và cập nhật thanh trạng thái của người chơi:

- **self.all\_sprites.update()**: Cập nhật trạng thái của tất cả các sprite trong trò chơi, bao gồm các khối, kẻ địch, vũ khí, viên đạn, và các sprite khác.
- **self.bars.update()**: Cập nhật thanh trạng thái của người chơi, bao gồm thanh máu, giáp, và năng lượng.

```
def draw(self):
    #game loop draw
    self.screen.fill(BLACK)
    self.all_sprites.draw(self.screen)
    #draw health and armor bar
    self.bars.draw(self.screen)
    self.clock.tick(FPS)
    pygame.display.update()
```

Phương thức **draw** trong class **Game** làm nhiệm vụ vẽ tất cả các sprite lên màn hình:

- **self.screen.fill(BLACK)**: Xóa màn hình bằng màu đen.
- **self.all\_sprites.draw(self.screen)**: Vẽ tất cả các sprite trong trò chơi lên màn hình.
- **self.bars.draw(self.screen)**: Vẽ thanh trạng thái của người chơi (thanh máu, giáp và năng lượng) lên màn hình.
- **self.clock.tick(FPS)**: Đảm bảo tần số khung hình đạt được FPS (khung hình mỗi giây).
- **pygame.display.update()**: Cập nhật màn hình với các thay đổi vẽ mới.

```
def main(self):
    #game loop
    while(self.playing):
        self.events()
        self.update()
        if self.maps.check_win():
            self.playing = False
        self.draw()
```

Phương thức **main** trong class **Game** là vòng lặp chính của trò chơi:

- Trong vòng lặp **while(self.playing)**, trò chơi sẽ tiếp tục chạy khi biến **self.playing** được đặt thành **True**.
- **self.events()**: Xử lý các sự kiện trong game như bấm phím, click chuột hoặc thoát game.
- **self.update()**: Cập nhật trạng thái của tất cả các sprite trong trò chơi.
- **if self.maps.check\_win()**: Kiểm tra xem người chơi đã chiến thắng trò chơi chưa. Nếu có, biến **self.playing** sẽ được đặt thành **False**, kết thúc vòng lặp chính.
- **self.draw()**: Vẽ tất cả các sprite lên màn hình.

```

def gameover(self):
    text = self.font.render('Game Over', True, RED)
    text_rect = text.get_rect(center=(WIN_WIDTH/2, WIN_HEIGHT/2))

    restart_button = Button(10, 50, 120, 50, WHITE, BLACK, 'Restart', 32)

    for sprite in self.all_sprites:
        sprite.kill()

    while(self.running):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False

        mouse_pos = pygame.mouse.get_pos()
        mouse_pressed = pygame.mouse.get_pressed()
        if restart_button.isPressed(mouse_pos, mouse_pressed):
            self.new()
            self.main()

        self.screen.blit(self.gameover_background, (0,0))
        self.screen.blit(text, text_rect)
        self.screen.blit(restart_button.image, restart_button.rect)
        self.clock.tick(FPS)
        pygame.display.update()
    pass

```

Phương thức **gameover** trong class **Game** được sử dụng để hiển thị màn hình kết thúc game và xử lý việc khởi động lại trò chơi sau khi game over:

- Đầu tiên, nó tạo một đối tượng văn bản **text** để hiển thị chữ "Game Over" ở giữa màn hình.
- Sau đó, nó tạo một nút khởi động lại bằng cách tạo một đối tượng **Button**.
- Tiếp theo, nó loại bỏ tất cả các sprite khỏi tất cả các nhóm sprite.
- Trong vòng lặp **while(self.running)**, nó kiểm tra các sự kiện như bấm phím hoặc click chuột. Nếu người chơi bấm vào nút khởi động lại, nó sẽ gọi phương thức **new** để bắt đầu một trò chơi mới, và sau đó gọi phương thức **main** để bắt đầu vòng lặp chính của trò chơi.
- Cuối cùng, nó vẽ màn hình game over, bao gồm nền và các nút khởi động lại và văn bản "Game Over".

```

def win_screen(self):
    outro = True
    title = self.font.render('Pygame RPG', True, BLACK)
    title_rect = title.get_rect(x=10,y=10)
    text = self.font.render('You win!!!', True, BLUE)

```

```

text_rect = text.get_rect(center=(WIN_WIDTH/2, WIN_HEIGHT/2))

play_button = Button(10, 50, 120, 50, WHITE, GREY, 'Play', 32)
exit_button = Button(10, 120, 120, 50, WHITE, RED, 'Exit', 32)
while(outro):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.running = False
            outro = False

    mouse_pos = pygame.mouse.get_pos()
    mouse_pressed = pygame.mouse.get_pressed()
    if play_button.isPressed(mouse_pos, mouse_pressed):
        outro = False
    if exit_button.isPressed(mouse_pos, mouse_pressed):
        self.running = False
        outro = False
    self.screen.blit(self.intro_background, (0,0))
    self.screen.blit(title, title_rect)
    self.screen.blit(play_button.image, play_button.rect)
    self.screen.blit(exit_button.image, exit_button.rect)
    self.screen.blit(text, text_rect)
    self.clock.tick(FPS)
    pygame.display.update()

if self.running:
    self.new()

```

Phương thức **win\_screen** trong class **Game** được sử dụng để hiển thị màn hình chiến thắng khi người chơi chiến thắng trò chơi:

- Đầu tiên, nó tạo các đối tượng văn bản để hiển thị tiêu đề "Pygame RPG" và thông báo "You win!!!".
- Sau đó, nó tạo hai nút: một nút "Play" và một nút "Exit" bằng cách tạo các đối tượng **Button**.
- Trong vòng lặp **while(outro)**, nó kiểm tra các sự kiện như bấm phím hoặc click chuột. Nếu người chơi bấm vào nút "Play", biến **outro** được đặt thành False và vòng lặp kết thúc. Nếu người chơi bấm vào nút "Exit", biến **outro** và **self.running** đều được đặt thành False, kết thúc vòng lặp và kết thúc trò chơi.
- Trong khi vòng lặp **outro** đang chạy, nó vẽ màn hình chiến thắng, bao gồm nền, tiêu đề, các nút "Play" và "Exit" cùng với văn bản "You win!!!".
- Nếu trò chơi vẫn đang chạy (biến **self.running** là True), nó gọi phương thức **new** để bắt đầu một trò chơi mới.

```
def intro_screen(self):
```

```

intro = True
title = self.font.render('Pygame RPG', True, BLACK)
title_rect = title.get_rect(x=10,y=10)
text = self.font.render('Let\'s Play', True, RED)
text_rect = text.get_rect(center=(WIN_WIDTH/2, WIN_HEIGHT/2))

play_button = Button(10, 50, 120, 50, WHITE, BLACK, 'Play', 32)

while(intro):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.running = False
            intro = False

        mouse_pos = pygame.mouse.get_pos()
        mouse_pressed = pygame.mouse.get_pressed()
        if play_button.isPressed(mouse_pos, mouse_pressed):
            intro = False

    self.screen.blit(self.intro_background, (0,0))
    self.screen.blit(title, title_rect)
    self.screen.blit(play_button.image, play_button.rect)
    self.screen.blit(text, text_rect)
    self.clock.tick(FPS)
    pygame.display.update()
pass

```

Phương thức **intro\_screen** trong class **Game** được sử dụng để hiển thị màn hình giới thiệu trò chơi khi trò chơi được khởi động:

- Đầu tiên, nó tạo các đối tượng văn bản để hiển thị tiêu đề "Pygame RPG" và thông báo "Let's Play".
- Sau đó, nó tạo một nút "Play" bằng cách tạo một đối tượng **Button**.
- Trong vòng lặp **while(intro)**, nó kiểm tra các sự kiện như bấm phím hoặc click chuột. Nếu người chơi bấm vào nút "Play", biến **intro** được đặt thành False và vòng lặp kết thúc.
- Trong khi vòng lặp **intro** đang chạy, nó vẽ màn hình giới thiệu, bao gồm nền, tiêu đề, nút "Play" cùng với văn bản "Let's Play".
- Sau khi người chơi bấm vào nút "Play", vòng lặp kết thúc và trò chơi bắt đầu.

```

def main():
    g = Game()
    g.intro_screen()
    g.new()
    while(g.running):
        g.main()
        if g.maps.check_win():
            g.win_screen()

```



```

        else: g.gameover()

    pygame.quit()
    sys.exit()

if __name__ == "__main__":
    main()

```

Trong hàm **main()**:

- Đầu tiên, một đối tượng **Game** mới được tạo và lớp **intro\_screen()** được gọi để hiển thị màn hình giới thiệu.
- Sau đó, trò chơi mới được khởi tạo bằng cách gọi phương thức **new()** của đối tượng **Game**.
- Trong vòng lặp **while(g.running)**, phương thức **main()** của đối tượng **Game** được gọi liên tục để thực hiện vòng lặp chính của trò chơi. Nếu đối tượng **MapList** kiểm tra thắng cuộc thông qua phương thức **check\_win()**, màn hình chiến thắng sẽ được hiển thị thông qua phương thức **win\_screen()**. Nếu không, màn hình thất bại sẽ được hiển thị thông qua phương thức **gameover()**.
- Khi vòng lặp kết thúc, thư viện Pygame sẽ được dọn dẹp thông qua **pygame.quit()**, và hàm **sys.exit()** được gọi để thoát khỏi chương trình.
- Điều kiện **if \_\_name\_\_ == "\_\_main\_\_":** đảm bảo rằng **main()** chỉ được gọi khi mã chạy là mã chính của chương trình.

#### 4. 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 Dungeon World. 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.

