

## A. Các lớp dùng để tạo Tiles:

### 1. import

```
1 from support import import_folder
2 import pygame
```

- **from support import import\_folder** dùng để import hàm **import\_folder** từ file support.py. Hàm **import\_folder** được dùng lấy các hình ảnh từ thư mục chỉ định và trả về 1 danh sách các hình ảnh được sử dụng trong việc tạo và hiển thị các hình ảnh lên màn hình.

### 2. Lớp Tile

```
1 class Tile(pygame.sprite.Sprite):
2     def __init__(self, size, x, y):
3         super().__init__()
4         self.image = pygame.Surface((size, size)) # Tạo một bề mặt (Surface) với kích thước (size, size)
5         self.rect = self.image.get_rect(topleft=(x, y)) # Lấy hình chữ nhật (rectangle) bao quanh bề mặt và đặt vị trí cho nó
6
7     def update(self, shift):
8         self.rect.x += shift
```

- Lớp **Tile** kế thừa từ **pygame.sprite.Sprite** định nghĩa một **sprite Pygame**. Được dùng để tạo ra và quản lý các tile trên màn hình trò chơi.

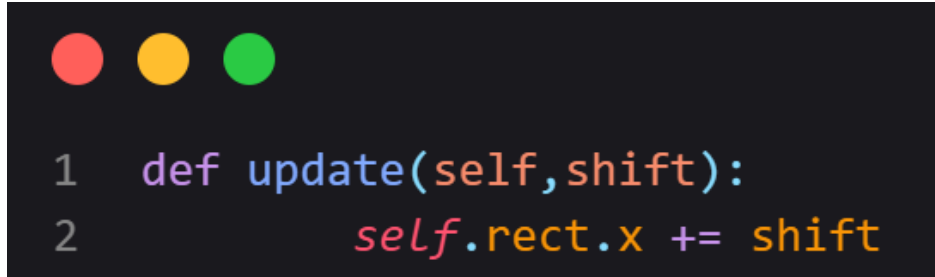
#### 2.1: Phương thức khởi tạo **\_\_init\_\_((self, size, x, y))**

```
1 def __init__(self, size, x, y):
2     super().__init__()
3     self.image = pygame.Surface((size, size))
4     self.rect = self.image.get_rect(topleft=(x, y))
5
```

- Phương thức **\_\_init\_\_** được gọi khi tạo một đối tượng mới của lớp **Tile**. Trong đó:
  - **size, x, y**: lần lượt là kích thước của 1 ô và tọa độ x và y của ô trên màn hình.

- **self.image**: Là một bề mặt Pygame được tạo ra với kích thước **size \* size**. Tạo ra ô có hình dạng là hình vuông với kích thước đã cho.
- **self.rect**: Là hình chữ nhật bao quanh bề mặt (**self.image**). **get\_rect()** trả về một hình chữ nhật có kích thước bằng với bề mặt và vị trí (**x, y**) được chỉ định bởi tham số **opleft**.

## 2.2: Phương thức **update(self, shift)**



```

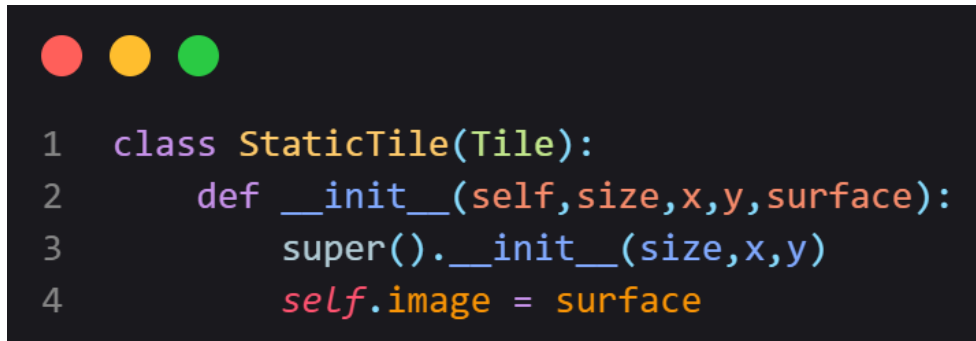
1  def update(self, shift):
2      self.rect.x += shift

```

- Phương thức **update** trên có chức năng cập nhật vị trí của ô trên màn hình dựa trên một giá trị dịch chuyển (shift) đã cho trước.
- Cụ thể:
  - Tham số **shift**: là một giá trị số nguyên biểu thị cho số lượng pixel cần di chuyển ô theo trục x.
  - **self.rect.x += shift**: cập nhật vị trí của tất cả các ô trên màn hình dựa trên việc di chuyển của người chơi hoặc các sự kiện khác

Tóm lại khi gọi phương thức update của lớp Tile ta truyền vào nó 1 giá trị shift một cách liên tục từ các hàm xử lý sự kiện trong class Level giúp cập nhật vị trí của tất cả các ô trên màn hình sang phải (shift dương) hoặc trái (shift âm) dựa trên việc di chuyển của người chơi.

## 3.Lớp **StaticTile**



```

1  class StaticTile(Tile):
2      def __init__(self, size, x, y, surface):
3          super().__init__(size, x, y)
4          self.image = surface

```

- Lớp **StaticTile** là lớp con của **Tile** dùng để tạo ra các ô cố định trên màn hình mà có hình ảnh không thay đổi. Ví dụ như: mặt đất, cây cối, thùng, ...
- Phương thức `__init__` :
  - o Tham số:
    - Size: kích thước của 1 ô.
    - x,y: tọa độ x,y của ô trên màn hình.
    - surface: sử dụng để chứa hình ảnh được sử dụng để hiển thị cho ô đó trên màn hình.
  - o Gán thuộc tính `self.image = surface` khi ô này được vẽ lên màn hình nó sẽ hiển thị hình ảnh được gán(surface).

#### 4.Lớp Crate

```

1 class Crate(StaticTile):
2     def __init__(self,size,x,y):
3         super().__init__(size,x,y,pygame.image.load('graphics/terrain/crate.png').convert_alpha())
4         offset_y = y + size
5         self.rect = self.image.get_rect(bottomleft = (x,offset_y))

```

Lớp Crate kế thừa từ lớp StaticTile dùng để tạo đối tượng là 1 cái thùng gỗ (crate).

##### 4.1: Phương thức `__init__`

- Gọi lại phương thức `__init__` của lớp cha **StaticTile** với các tham số **size, x, y** và hình ảnh của thùng (**crate.png**) thay cho surface.
- Do khi hiển thị đối tượng thùng (crate) lên màn hình thì đối tượng được hiển thị lơ lửng trên không và không trên mặt đất (terrain) nên ta phải xử lý lại vị trí của trục y của thùng bằng cách gán vị trí mới của thùng là: **offset\_y = y + size** và dùng đoạn mã **self.rect = self.image.get\_rect(bottomleft=(x, offset\_y))** để thiết lập vị trí của hình chữ nhật bao quanh hình ảnh của thùng (crate) sao cho góc dưới bên trái của hình ảnh đúng là (**x, offset\_y**).

#### 5.Lớp AnimateTile

```

1  class AnimateTile(Tile):
2      def __init__(self,size,x,y,path):
3          super().__init__(size,x,y)
4          self.frames = import_folder(path)
5          self.frame_index = 0
6          self.image = self.frames[self.frame_index]
7
8      def animate(self):
9          self.frame_index += 0.15
10         if self.frame_index >= len(self.frames):
11             self.frame_index = 0
12             self.image = self.frames[int(self.frame_index)]
13
14     def update(self,shift):
15         self.animate()
16         self.rect.x += shift

```

Lớp **AnimateTile** kế thừa từ lớp **Tile**, lớp **AnimateTile** dùng để tạo ra các ô hoạt ảnh. Ví dụ như animation của enemy,player, của cây cối,....

### 5.1: Phương thức `__init__`

```

def __init__(self,size,x,y,path):
    super().__init__(size,x,y)
    self.frames = import_folder(path)
    self.frame_index = 0
    self.image = self.frames[self.frame_index]

```

- Có các tham số **size, x, y** với các chức năng như trong lớp cha **Tile** và biến **path** dùng để chứa đường dẫn tới thư mục chứa các frames của đối tượng cần tạo hoạt ảnh.
- Tiếp theo gọi lại phương thức của lớp cha để thiết lập các thuộc tính cơ bản cho ô đó.
- Dòng **self.frames = import\_folder(path)** để lưu danh sách các hình ảnh (frames) của đối tượng cần tạo hoạt ảnh.
- Dòng **self.frame\_index = 0** được dùng để xác định chỉ số của frames hiện tại mà lớp **AnimateTile** đang hiển thị.
- Dòng **self.image = self.frames[self.frame\_index]** gán thuộc tính hình ảnh (**self.image**) là frame đầu tiên trong danh sách **self.frames**

## 5.2: Phương thức **animate**

```
def animate(self):
    self.frame_index += 0.15
    if self.frame_index >= len(self.frames):
        self.frame_index = 0
    self.image = self.frames[int(self.frame_index)]
```

Phương thức chuyển đổi giữa các frame hoạt ảnh để tạo hiệu ứng hoạt ảnh cho các ô.

- **self.frame\_index += 0.15**: Mỗi lần phương thức **animate** được gọi **self.frame\_index** sẽ tăng thêm **0.15** nó sẽ giúp tạo ra hiệu ứng chuyển mịn màng hơn khi chuyển đổi giữa các frame.
- Đoạn **if self.frame\_index >= len(self.frames)** có tác dụng khi Nếu chỉ số frame vượt quá số lượng frame có sẵn trong danh sách **self.frames**. Nghĩa là đã hiển thị hết các frame, ta sẽ thiết lập lại **self.frame\_index** về 0 làm cho hoạt ảnh lặp lại từ đầu, giúp tạo ra hiệu ứng hoạt ảnh liên tục và không giới hạn trong lớp **AnimateTile**.
- **self.image = self.frames[int(self.frame\_index)]**: Sau khi cập nhật **self.frame\_index**, ta sử dụng nó để chọn frame tương ứng từ danh sách **self.frames** và gán frame đó vào thuộc tính **self.image**. Nó sẽ hiển thị hình ảnh của frame tương ứng mỗi khi hàm **animate** được gọi.

## 5.3: Phương thức **update**

```
def update(self, shift):
    self.animate()
    self.rect.x += shift
```

Phương thức **update** trên dùng để cập nhật vị trí các đối tượng hoạt ảnh trên màn hình và gọi phương thức **self.animate()** để chuyển đổi giữa các frame ảnh tạo các hiệu ứng hoạt ảnh cho đối tượng hoạt ảnh động.

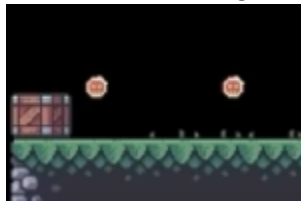
- **self.animate()**: phương thức **animate** để chuyển đổi giữa các **frame** hoạt ảnh và cập nhật hình ảnh của ô để hiển thị frame mới.
- **self.rect.x += shift**: Đoạn code sẽ thực hiện việc di chuyển đối tượng sang phải (**shift** là số dương) hoặc sang trái (**shift** là số âm) trên màn hình.

## 6.Lớp **Coin**

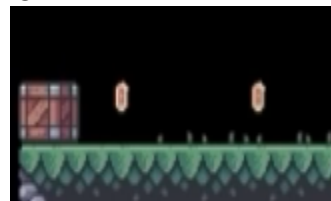
```
class Coin(AnimateTile):
    def __init__(self,size,x,y,path,value):
        super().__init__(size,x,y,path)
        center_x = x + int(size/2)
        center_y = y + int(size/2)
        self.rect = self.image.get_rect(center=(center_x,center_y))
        self.value = value
```

Lớp **Coin** kế thừa từ lớp **AnimateTile** được sử dụng để tạo ra các đối tượng đồng xu trong trò chơi, với khả năng hiển thị hoạt ảnh và có giá trị cụ thể.

- Phương thức **\_\_init\_\_ (self,size,x,y,path,value):**
  - Gọi phương thức **\_\_init\_\_** của lớp cha (**AnimateTile**) với các tham số **size**, **x**, **y**, và **path** để khởi tạo các thuộc tính cơ bản của đồng xu.
  - Tham số:
    - **size, x, y:** Kích thước, tọa độ x và tọa độ y của đồng xu.
    - **path:** Đường dẫn đến thư mục chứa các frame hoạt ảnh cho đồng xu.
    - **value:** Giá trị điểm của từng loại đồng xu (xu bạc 1 điểm, xu vàng 2 điểm).
  - **center\_x = x + int(size/2):** Dòng này để tính lại tọa độ x cho tâm của đồng xu bằng cách lấy tọa độ x của đồng xu cộng thêm **int(size/2)** để cho tọa độ x của đồng xu nằm ở giữa ô.
  - **center\_y = y + int(size/2):** Tương tự như trên, dòng này tính toán tọa độ y của tâm của đồng xu.
  - **self.rect = self.image.get\_rect(center=(center\_x, center\_y)):** đặt lại vị trí cho đồng xu vào vị trí trung tâm của ô



xu trước khi sửa tọa độ



xu sau khi sửa tọa độ

- **self.value = value:** dòng này gán giá trị đồng xu vào thuộc tính giá trị của đồng xu để sau này sẽ được sử dụng khi vào trò chơi để tính điểm xu

## 7. Phương thức **Palm:**

```
class Palm(AnimateTile):
    def __init__(self, size, x, y, path, offset):
        super().__init__(size, x, y, path)
        offset_y = y - offset
        self.rect.topleft = (x, offset_y)
```

- Lớp **Palm** được sử dụng để tạo ra các đối tượng cây cọ trong trò chơi, có khả năng hiển thị hoạt ảnh của cây cọ.
- Do khi đưa hoạt ảnh của cây cọ (Palm) vào màn hình trò chơi cây cọ bị ở quá sâu so với mặt đất nên đoạn code **offset\_y = y - offset** điều chỉnh lại vị trí cây cọ cho hợp lý với mặt đất và cuối cùng đoạn code **self.rect.topleft = (x, offset\_y)** sẽ đặt lại vị trí chính xác cho cây cọ.

Dưới đây là ảnh minh họa:



cây cọ trước khi thay đổi  
vị trí theo trục y



cây cọ sau khi thay đổi  
vị trí theo trục y

## B. moving\_terrain

### 1. Lớp Moving\_Terrain

```
1 class Moving_Terrain(pygame.sprite.Sprite):
2     def __init__(self, width, height, x, y, surface):
3         super().__init__()
4         self.image = pygame.Surface((width, height)) # Tạo một bề mặt (Surface) với kích thước (size, size)
5         self.rect = self.image.get_rect(topleft=(x, y))
6         self.image = surface
7         self.speed = 3
8         self.direction = pygame.math.Vector2(0, 0)
9
10    def move(self):
11        self.rect.x += self.speed
12
13    def reverse(self):
14        self.direction.x = -1
15        self.speed *= self.direction.x
16
17    def update(self, shift):
18        self.rect.x += shift
19        self.move()
```

Lớp **Moving\_Terrain** trên được dùng để khởi tạo ra một đối tượng địa hình lơ lửng có thể di chuyển qua lại trong một khoảng nhất định theo trục x để kết nối hai địa hình ở xa nhau.



(ảnh minh họa)

### 1.1: Phương thức khởi tạo `__init__` của class **Moving\_Terrain**

```
1 def __init__(self, width,height, x, y,surface):
2     super().__init__()
3     self.image = pygame.Surface((width,height))# Tạo một bề mặt (Surface) với kích thước (size, size)
4     self.rect = self.image.get_rect(topleft=(x,y))
5     self.image = surface
6     self.speed = 3
7     self.direction = pygame.math.Vector2(0,0)
```

- Phương thức `__init__` tham số:
  - o **width, height**: Chiều rộng và chiều cao của đối tượng.
  - o **x, y**: Là tọa độ x và y ban đầu của đối tượng trên màn hình.
  - o **surface**: hình ảnh của đối tượng.
- **self.image = pygame.Surface((width, height))**: Tạo ra một bề mặt có kích thước là width,height để hiển thị hình ảnh của đối tượng.
- **self.rect = self.image.get\_rect(topleft=(x, y))**: Tạo một hình chữ nhật bao quanh đối tượng nó sẽ giúp xử lý va chạm và xác định vị trí và kích thước của đối tượng dễ hơn.
- **self.image = surface**: Gán hình ảnh surface cho thuộc tính self.image để có thể vẽ hình ảnh lên vị trí được xác định từ trước.
- **self.speed = 3**: Khởi tạo tốc độ di chuyển cho đối tượng là 3 pixel sau mỗi khung hình.
- **self.direction = pygame.math.Vector2(0, 0)**: Khởi tạo vector self.direction xác định hướng di chuyển của nhân vật với giá trị ban đầu là (0, 0) – đối tượng đứng yên chưa di chuyển.



## 1.2: Phương thức **move**

```
def move(self):  
    self.rect.x += self.speed
```

Dùng để di chuyển đối tượng

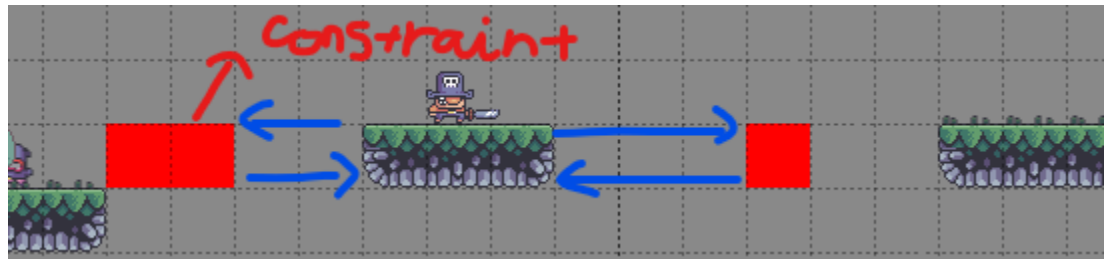
- **self.rect.x += self.speed** : dòng code dùng để di chuyển đối tượng địa hình theo trục x với tốc độ **self.speed**.

## 1.3: Phương thức **reverse**

```
def reverse(self):  
    self.direction.x = -1  
    self.speed *= self.direction.x
```

Dùng để đảo chiều di chuyển cho đối tượng địa hình

- **self.direction.x = -1**: Gán thuộc tính **direction** trục x bằng -1 khi để đổi quay ngược hướng di chuyển hiện tại của đối tượng.
- **self.speed \*= self.direction.x**: Đoạn code làm cho đối tượng di chuyển ngược lại khi hướng di chuyển thay đổi.
  - Ví dụ: khi đối tượng chạm với hai đối tượng constraint vô hình đối tượng địa hình sẽ đổi hướng và di chuyển theo chiều ngược lại so với trước khi va chạm.



(ảnh minh họa)

## 1.4: Phương thức **update**

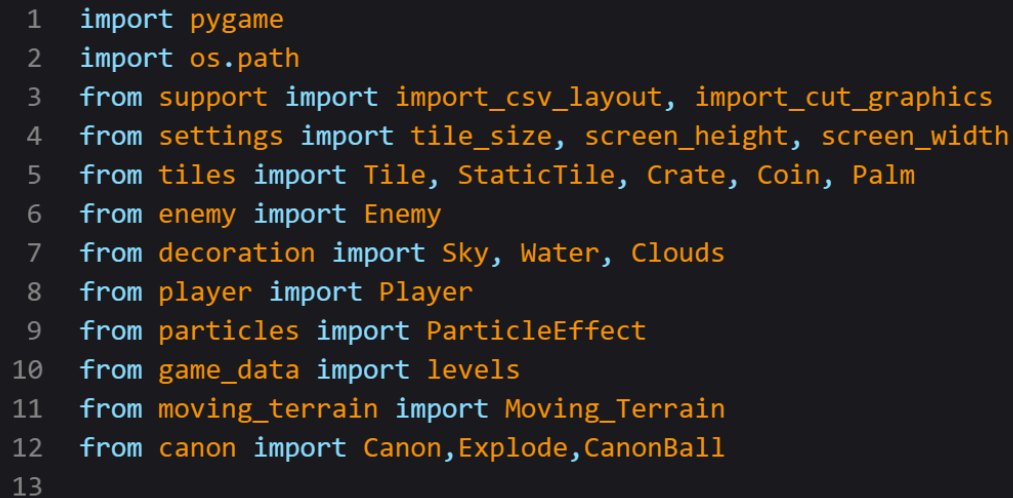
Phương thức **update** di chuyển đối tượng theo tốc độ dịch chuyển của khung hình và cho đối tượng khả năng di chuyển.

```
def update(self, shift):  
    self.rect.x += shift  
    self.move()
```

- **self.rect.x += shift**: Dùng để di chuyển đối tượng theo tốc độ di chuyển của khung hình.
- **Self.move()**: Gọi phương thức **move()** để cho đối tượng khả năng di chuyển.

## C.level

### 1. Import module



```

1  import pygame
2  import os.path
3  from support import import_csv_layout, import_cut_graphics
4  from settings import tile_size, screen_height, screen_width
5  from tiles import Tile, StaticTile, Crate, Coin, Palm
6  from enemy import Enemy
7  from decoration import Sky, Water, Clouds
8  from player import Player
9  from particles import ParticleEffect
10 from game_data import levels
11 from moving_terrain import Moving_Terrain
12 from canon import Canon,Explode,CanonBall
13

```

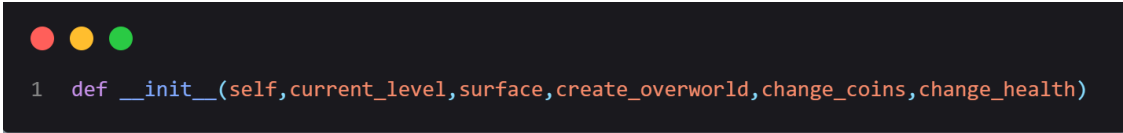
Ở đây ta sẽ import các module từ các tệp từ các tệp cục bộ trong dự án dùng cho việc khởi tạo các đối tượng khác nhau cho từng level của trò chơi và các module từ thư viện của python dùng để làm việc với tính năng đồ họa (pygame) và quản lý tệp(os.path)

### 2.Lớp Level

Lớp **Level** này được dùng để quản lý và hiển thị các đối tượng trong các level của trò chơi, bao gồm khởi tạo và tải dữ liệu, xử lý va chạm giữa các đối tượng với nhau, quản lý việc di chuyển và hiển thị các đối tượng, kiểm tra trạng thái của trò chơi xem người chơi đã thắng hay thất bại.

#### 2.1:Phương thức \_\_init\_\_ trong lớp level

- Các Tham số:



```

1  def __init__(self,current_level,surface,create_overworld,change_coins,change_health)

```

- **current\_level:** Tham số này sẽ xác định cấp độ hiện tại của trò chơi, xác định dữ liệu và cài đặt cụ thể cho từng level trò chơi.
- **surface:** Tham số là bề mặt của pygame nơi các level được vẽ lên.



(ảnh minh họa)

- **create\_overworld:** Là một phương thức dùng để hiển thị, kết nối các màn đại diện cho từng level của trò chơi và hiển thị các level sau khi level trước được hoàn thành, khi được mở các level được tô đen.



(ảnh minh họa)

- **change\_coins:** Là phương thức dùng để cập nhật số điểm số khi người chơi ăn được xu nó sẽ lưu trữ và hiển thị lên màn hình trò chơi.
- **change\_health:** Là phương thức dùng để cập nhật số lượng máu còn lại của nhân vật và hiển thị lên thanh máu trong màn hình trò chơi.

- **Khởi tạo thuộc tính cơ bản:**

```
#general setup
self.display_surface = surface#Lưu trữ bề mặt hiển thị của level chơi.
self.world_shift = 0
```

Đầu tiên khởi tạo thuộc tính `self.display_surface = surface` để lưu trữ bề mặt hiển thị level. Tiếp theo khởi tạo thuộc tính `self.world_shift = 0` để thay đổi vị trí các sprite trong level khi di chuyển camera.

- **Tạo các đối tượng âm thanh sử dụng trong trò chơi**

```
#audio all
self.coins_sound = pygame.mixer.Sound('audio/effects/coin.wav')
self.stomp_sound = pygame.mixer.Sound('audio/effects/stomp.wav')
```

- **Kết Nối overworld và level:**

```
#Overworld connection
self.create_overworld = create_overworld
self.current_level = current_level
level_data = levels[self.current_level]
self.new_max_level = level_data['unlock']
```

- Gán thuộc tính `self.create_overworld` cho `create_overworld` để tạo ra màn chơi tiếp theo hoặc chuyển đến màn chơi khác khi cần thiết.
- Gán giá trị của biến `current_level` cho thuộc tính `current_level` thể hiện cho màn chơi hiện tại của người chơi.
- `level_data = levels[self.current_level]` : Lấy dữ liệu của màn chơi hiện tại từ từ điển `levels`.
- Cuối cùng dùng dòng lệnh `self.new_max_level = level_data['unlock']` để lấy giá trị đại diện cho màn chơi tiếp theo trong level hiện tại.

- **Khởi tạo các thuộc tính liên quan đến nhân vật**

```
#player
player_layout = import_csv_layout(level_data['player'])
self.player = pygame.sprite.GroupSingle()
self.goal = pygame.sprite.GroupSingle()
self.setup_player(player_layout, change_health)
```

- Lấy vị trí xuất phát và vị trí của sprite mục tiêu cần đạt được để qua màn tiếp theo của nhân vật và gán cho biến `player_layout`.

- Khởi tạo thuộc tính **self.player** là 1 nhóm các sprite đại diện và hiển thị hình ảnh của nhân vật.
  - Tạo thuộc tính **self.goat** để chứa sprite về vị trí cần đạt được để qua màn của nhân vật.
  - Cuối cùng gọi phương thức **self.setup\_player** và truyền vào các đối số **player\_layout** và **change\_health** để tạo và cấu hình nhân vật trong trò chơi.
- **Tạo thuộc gọi phương thức **change\_coins** để cập nhật điểm số khi nhân vật ăn được xu và gán nó cho thuộc tính **self.change\_coins****

```
#user interface
self.change_coins = change_coins
```

- **Tạo các thuộc tính liên quan tới nhân vật**

```
#particles
#dust
self.dust_sprite = pygame.sprite.GroupSingle()
self.player_on_ground = False
#explosion
self.explosion_sprites = pygame.sprite.Group()
```

- Tạo **self.dust\_sprite** là nhóm các sprite hiển thị các hiệu ứng bụi khi nhân vật chuyển động.
  - Tạo thuộc tính **self.player\_on\_ground** để xác định xem nhân vật có đang đứng trên mặt đất hay không để tạo các hiệu ứng bụi.
  - Cuối cùng là thuộc tính **self.explosion\_sprites** dùng cho để hiển thị hiệu ứng nổ khi nhân vật tiêu diệt được 1 con quái.
- **Tạo các thuộc tính cho việc hiển thị các đối tượng của 1 level lên màn hình như cây cọ, địa hình, quái,....**

```

1  # terrain setup
2  terrain_layout = import_csv_layout(level_data['terrain'])
3  self.terrain_sprites = self.create_tile_group(terrain_layout, 'terrain') #Lưu trữ 1 nhóm sprite
4
5  #terrain moving
6  terrain_file = level_data.get('moving_terrain')
7
8  if terrain_file and os.path.exists(terrain_file):
9      moving_terrain_layout = import_csv_layout(level_data['moving_terrain'])
10     self.terrain_moving_sprites = pygame.sprite.Group()
11     self.setup_terrain_moving(moving_terrain_layout)
12 else:
13     self.terrain_moving_sprites = pygame.sprite.Group()
14
15 #setup canon
16 self.canon_file = level_data.get('canon')
17 #khởi tạo nhóm sprite cho đạn canon
18 self.canonball_sprites = pygame.sprite.Group()
19
20 if self.canon_file and os.path.exists(self.canon_file):
21
22     canon_layout = import_csv_layout(level_data['canon'])
23     self.canon_sprites = self.create_tile_group(canon_layout, 'canon')
24
25     canonball_layout = import_csv_layout(level_data['canonball'])
26     self.canonball_sprites = self.create_tile_group(canonball_layout, 'canonball')
27
28     explode = import_csv_layout(level_data['explode'])
29     self.explode_sprites = self.create_tile_group(explode, 'explode')

```

```

1  #grass layout
2  grass_layout = import_csv_layout(level_data['grass']) #Lấy dữ liệu tọa độ level từ file csv sử dụng hàm import
3  self.grass_sprites = self.create_tile_group(grass_layout, 'grass') #Lưu trữ 1 nhóm sprite
4  #coins
5  coins_layout = import_csv_layout(level_data['coins'])
6  self.coins_sprites = self.create_tile_group(coins_layout, 'coins') #Lưu trữ 1 nhóm sprite
7
8  #crates
9  crates_layout = import_csv_layout(level_data['crates'])
10 self.crates_sprites = self.create_tile_group(crates_layout, 'crates') #Lưu trữ 1 nhóm sprite
11
12 # fg palm
13 fg_palm_layout = import_csv_layout(level_data['fg palms'])
14 self.fg_palm_sprites = self.create_tile_group(fg_palm_layout, 'fg palms') #Lưu trữ 1 nhóm sprite
15
16 # bg palm
17 bg_palm_layout = import_csv_layout(level_data['bg palms'])
18 self.bg_palm_sprites = self.create_tile_group(bg_palm_layout, 'bg palms') #Lưu trữ 1 nhóm sprite
19
20 #enemy
21 enemies_layout = import_csv_layout(level_data['enemies'])
22 self.enemies_sprites = self.create_tile_group(enemies_layout, 'enemies') #Lưu trữ 1 nhóm sprite
23
24 #constraints
25 constraints_layout = import_csv_layout(level_data['constraints'])
26 self.constraints_sprites = self.create_tile_group(constraints_layout, 'constraints') #Lưu trữ 1 nhóm sprite
27

```

Tạo các biến để lưu vị trí của các đối tượng thông qua file csv rồi gọi phương thức ***self.create\_tile\_group*** dùng để lấy sprite tương ứng của từng đối tượng và gán nó cho thuộc tính sprite tương ứng của đối tượng đó. Ví dụ: ***self.enemies\_sprites = self.create\_tile\_group(enemies\_layout, 'enemies')*** trong đó ***self.enemies\_sprite*** là thuộc tính dùng lưu trữ sprite của đối tượng enemy.

- **Tạo các thuộc tính dùng để trang trí**

```
1  #decoration
2  self.sky = Sky(8)
3  level_width = len(terrain_layout[0]) * tile_size
4  self.water = Water(screen_height - 40, level_width)
5  self.clouds = Clouds(300, level_width, 35)
6
```

Tạo các thuộc tính để hiển thị các đối tượng trang trí level lên mà hình như là mây nước và bầu trời. Riêng *level\_width = len(terrain\_layout[0]) \* tile\_size* dùng để tính độ rộng của màn chơi hiện tại sử dụng trong việc hiển thị các đối tượng trang trí.

- **Các thuộc tính dùng để hiển thị 1 đoạn message**

```
1  self.show_Message = False
2  self.message_start_time = pygame.time.get_ticks()
3  self.message_duration = 0
```

Các thuộc tính trên dùng để hiển thị đoạn tin nhắn khi mà nhân vật chưa ăn hết xu nhưng lại muốn qua màn tiếp theo.

## 2.2.Phương thức create\_tile\_group

```
1 def create_tile_group(self,layout,type):
2     sprite_group = pygame.sprite.Group()#Lớp giúp tổ chức và quản lí sprite
3     for row_index, row in enumerate(layout):
4         for col_index,val in enumerate(row):
5             if val != '-1':
6                 x = col_index * tile_size
7                 y = row_index * tile_size
8
9                 if type == 'terrain':
10                     terrain_tile_list = import_cut_graphics('graphics/terrain/terrain_tiles.png')
11                     tile_surface = terrain_tile_list[int(val)]
12                     sprite = StaticTile(tile_size,x,y,tile_surface)
13
14                 if type == 'grass':
15                     grass_tile_list = import_cut_graphics('graphics/decoration/grass/grass.png')
16                     tile_surface = grass_tile_list[int(val)]
17                     sprite = StaticTile(tile_size,x,y,tile_surface)
18
19                 if type == 'coins':
20                     if val == '0': sprite = Coin(tile_size,x,y,'graphics/coins/gold',2)
21                     if val == '1': sprite = Coin(tile_size,x,y,'graphics/coins/silver',1)
22
23                 if type == 'crates':
24                     sprite = Crate(tile_size,x,y)
25
26                 if type == 'fg palms':
27                     if val == '0': sprite = Palm(tile_size,x,y,'graphics/terrain/palm_small',38)
28                     if val == '1': sprite = Palm(tile_size,x,y,'graphics/terrain/palm_large',64)
29
30                 if type == 'bg palms':
31                     sprite = Palm(tile_size,x,y,'graphics/terrain/palm_bg',64)
32
33                 if type == 'enemies':
34                     sprite = Enemy(tile_size,x,y)
35
36                 if type == 'canon':
37                     canon = Canon(tile_size,x,y)
38                     sprite = canon
39                     canon.fire(self.canonball_sprites,5)
40
41                 if type == 'explode':
42                     sprite = Explode(tile_size,x,y)
43
44                 if type == 'canonball':
45                     sprite = CanonBall(tile_size, x, y) # Gán giá trị cho sprite
46
47                 if type == 'constraints':
48                     sprite = Tile(tile_size,x,y)
49
50                 sprite_group.add(sprite)
51     return sprite_group
```

Phương thức trên dùng để tạo các nhóm sprite cho từng đối tượng cụ thể

```
sprite_group = pygame.sprite.Group()#Lớp giúp tổ chức và quản
for row_index, row in enumerate(layout):
    for col_index,val in enumerate(row):
        if val != '-1':
            x = col_index * tile_size
            y = row_index * tile_size
```

Đoạn code trên khởi tạo nhóm sprite sau đó dùng 2 vòng lặp để duyệt qua từng ô trong *layout*. Vòng lặp đầu tiên duyệt qua từng hàng của layout, vòng lặp thứ 2 duyệt qua từng ô(cột) trong hàng hiện tại sau đó xét điều kiện khi giá trị của *val* khác **-1** có nghĩa là tại vị



trí đó là vị trí tồn tại 1 sprite nào đó ta sẽ tính tọa độ x, y theo tọa độ pixel để hiển thị các đối tượng chính xác lên màn chơi.

### -Tạo các sprite dựa trên giá trị của type và val

```
1         if type == 'terrain':
2             terrain_tile_list = import_cut_graphics('graphics/terrain/terrain_tiles.png')
3             tile_surface = terrain_tile_list[int(val)]
4             sprite = StaticTile(tile_size,x,y,tile_surface)
5
6         if type == 'grass':
7             grass_tile_list = import_cut_graphics('graphics/decoration/grass/grass.png')
8             tile_surface = grass_tile_list[int(val)]
9             sprite = StaticTile(tile_size,x,y,tile_surface)
10
11        if type == 'coins':
12            if val == '0': sprite = Coin(tile_size,x,y,'graphics/coins/gold',2)
13            if val == '1': sprite = Coin(tile_size,x,y,'graphics/coins/silver',1)
14
15        if type == 'crates':
16            sprite = Crate(tile_size,x,y)
17
18        if type == 'fg palms':
19            if val == '0': sprite = Palm(tile_size,x,y,'graphics/terrain/palm_small',38)
20            if val == '1': sprite = Palm(tile_size,x,y,'graphics/terrain/palm_large',64)
21
22        if type == 'bg palms':
23            sprite = Palm(tile_size,x,y,'graphics/terrain/palm_bg',64)
24
25        if type == 'enemies':
26            sprite = Enemy(tile_size,x,y)
27
28        if type == 'canon':
29            canon = Canon(tile_size,x,y)
30            sprite = canon
31            canon.fire(self.canonball_sprites,5)
32
33        if type == 'explode':
34            sprite = Explode(tile_size,x,y)
35
36        if type == 'canonball':
37            sprite = CanonBall(tile_size, x, y) # Gán giá trị cho sprite
38
39        if type == 'constraints':
40            sprite = Tile(tile_size,x,y)
41
42        sprite_group.add(sprite)
43    return sprite_group
```

Đầu tiên ta dựa vào điều kiện **type** để xác định xem sprite nào cần được tạo, sau đó tùy vào từng loại sprite ta dựa vào giá trị của **val** để tạo ra đúng sprite cụ thể cho từng giá trị **val** với các thông số kích thước, vị trí và hình ảnh. Cuối cùng thêm sprite được tạo ra vào nhóm sprite **sprite\_group** rồi trả về **sprite\_group** chứa tất cả các sprite trong vòng lặp.

## 2.3. Phương thức `setup_terrain_moving`

```
1 def setup_terrain_moving(self, layout):
2     sprite_width = 3 * tile_size
3     sprite_height = tile_size
4
5     for row_index, row in enumerate(layout):
6         for col_index, val in enumerate(row):
7             if val != '-1':
8                 x = col_index * tile_size
9                 y = row_index * tile_size
10
11                 terrain_moving_tile = pygame.image.load('graphics/terrain/terrain_moving.png').convert_alpha()
12                 sprite = Moving_Terrain(sprite_width, sprite_height, x, y, terrain_moving_tile)
13                 self.terrain_moving_sprites.add(sprite)
```

Phương thức `setup_terrain_moving` dùng để tạo các sprite cho đối tượng *terrain\_moving* - một dạng địa hình trong trò chơi có khả năng lơ lửng và di chuyển.

`sprite_width = 3 * tile_size` và `sprite_height = tile_size` hai dòng này tính chiều rộng và chiều cao cho đối tượng, chiều rộng bằng  $3 * \text{tile\_size}$  ( $\text{tile\_size} = 64$ ) = 192 pixel và chiều cao bằng 64 pixel.

```
for row_index, row in enumerate(layout):
    for col_index, val in enumerate(row):
        if val != '-1':
            x = col_index * tile_size
            y = row_index * tile_size

            terrain_moving_tile = pygame.image.load('graphics/terrain/terrain_moving.png').convert_alpha()
            sprite = Moving_Terrain(sprite_width, sprite_height, x, y, terrain_moving_tile)
            self.terrain_moving_sprites.add(sprite)
```

Vòng lặp trên tương tự như vòng lặp trong phương thức `create_tile_group`.

## 2.4. Phương thức `terrain_moving_collision_reverse`

```
def terrain_moving_collision_reverse(self):
    for moving_terrain in self.terrain_moving_sprites:
        colliding_constraints = pygame.sprite.spritecollide(moving_terrain, self.constraints_sprites, False)
        if colliding_constraints:
            moving_terrain.reverse()
```

Phương thức `terrain_moving_collision_reverse` dùng để đổi chiều di chuyển của đối tượng *Moving\_Terrain* khi xảy ra va chạm với đối tượng *constraint* là đối tượng tượng trưng cho giới hạn di chuyển.

Ta sẽ tạo vòng lặp để duyệt qua từng sprite của thuộc tính `self.terrain_moving_sprites` sau đó ta gọi hàm `pygame.sprite.spritecollide()` để kiểm tra va chạm giữa các sprite của `self.terrain_moving_sprites` với sprite của `self.constraints_sprites` và `False` để chỉ xác định va chạm mà không thay đổi trạng thái của các sprite. Khi điều kiện xảy ra va chạm đúng thì `moving_terrain` sẽ gọi phương thức `reverse()` của đối tượng *Moving\_terrain* để đảo chiều hướng di chuyển.

Ở đây đối tượng **constraint** chỉ là 1 đối tượng **Tile** dùng xác định ranh giới và không cần hiển thị hình ảnh lên màn hình.

## 2.5. Phương thức `setup_player`

```
1 def setup_player(self, layout, change_health):
2     for row_index, row in enumerate(layout):
3         for col_index, val in enumerate(row):
4             x = col_index * tile_size
5             y = row_index * tile_size
6             if val == '0':
7                 sprite = Player((x,y), self.display_surface, self.create_jump_particles, change_health)
8                 self.player.add(sprite)
9             if val == '1':
10                hat_surface = pygame.image.load('graphics/character/hat.png').convert_alpha()
11                sprite = StaticTile(tile_size, x, y, hat_surface)
12                self.goal.add(sprite)
```

Phương thức **setup\_player** được sử dụng để khởi tạo vị trí bắt đầu của nhân vật tại vị trí giá trị của **val** = **'0'** và sprite của mục tiêu cần chạm tới để kết thúc của màn.

```
if val == '0':
    sprite = Player((x,y), self.display_surface, self.create_jump_particles, change_health)
    self.player.add(sprite)
```

Đoạn code trên sẽ tạo sprite nhân vật tại tọa độ của ô có giá trị **val** == **'0'**, sau đó sprite được thêm vào **self.player** để quản lí nhân vật.

```
if val == '1':
    hat_surface = pygame.image.load('graphics/character/hat.png').convert_alpha()
    sprite = StaticTile(tile_size, x, y, hat_surface)
    self.goal.add(sprite)
```

Trong đoạn code trên, tải lên hình ảnh của đối tượng nhân vật cần chạm vào để kết thúc màn chơi sau đó gán nó cho biến **hat\_surface**, tiếp theo tạo đối tượng **StaticTile** với các đối số **tile\_size()** kích thước của ô, **x** và **y** là tọa độ pixel của ô và **hat\_surface** là hình ảnh của đối tượng. Cuối cùng là thêm sprite của đối tượng vào thuộc tính **self.goal** để hiển thị lên màn hình.



(ảnh màu cam là đối tượng nhân vật  
cần chạm vào để qua màn)

## 2.6. Phương thức `enemy_collision_reverse`

```
1 def enemy_collision_reverse(self):
2     for enemy in self.enemies_sprites.sprites():
3         if pygame.sprite.spritecollide(enemy, self.constraints_sprites, False):
4             enemy.reverse()
```

Phương thức trên cũng tương tự như phương thức *terrain\_moving\_collision\_reverse*, nó cũng đảo chiều hướng di chuyển của quái (enemy) khi xảy ra va chạm với các sprite của constraint.



Quái sẽ đổi hướng khi va chạm với constraint

## 2.7. Phương thức `create_jump_particles`

```
1 def create_jump_particles(self, pos):
2     if self.player.sprite.facing_right:
3         pos -= pygame.math.Vector2(10, 5)
4     else:
5         pos += pygame.math.Vector2(10, -5)
6     jump_particle_sprite = ParticleEffect(pos, 'jump')
7     self.dust_sprite.add(jump_particle_sprite)
```

Phương thức *create\_jump\_particles* dùng để tạo ra các hạt bụi khi nhân vật nhảy.

Dựa vào vị trí của nhân vật, nếu nhân vật nhảy về bên phải thì hạt bụi sẽ được hiển thị lệch sang bên trái một chút và ngược lại. Cuối cùng thêm sprite vào thuộc tính *self.dust\_sprite* để hiển thị lên màn hình khi xảy ra hành động nhảy của nhân vật.

## 2.8. Phương thức `horizontal_movement_collision`

```

1 def horizontal_movement_collision(self):
2     player = self.player.sprite
3     player.collision_rect.x += player.direction.x * player.speed
4     collidable_sprites = (self.terrain_sprites.sprites() + self.crates_sprites.sprites()
5                           + self.fg_palm_sprites.sprites() + self.terrain_moving_sprites.sprites())
6
7     for sprite in collidable_sprites:
8         if sprite.rect.colliderect(player.collision_rect):
9             if player.direction.x < 0:
10                 player.collision_rect.left = sprite.rect.right
11                 player.on_left = True
12             elif player.direction.x > 0:
13                 player.collision_rect.right = sprite.rect.left
14                 player.on_right = True

```

Phương thức trên được thiết kế để xử lý các va chạm của nhân vật với các vật thể theo phương ngang.

Đầu tiên gọi thuộc tính *self.player.sprite* và gán cho biến *player*, tiếp theo di chuyển vùng va chạm của nhân vật bằng với tốc độ và hướng hiện tại của nó (code dòng số 2), sau đó tạo một danh sách các sprite mà nhân vật có thể sẽ va chạm theo phương ngang.

Cuối cùng phương thức sẽ lặp qua từng sprite và kiểm tra xem có xảy ra va chạm giữa nhân vật và sprite đó hay không bằng phương thức *colliderect*, nếu có va chạm và nhân vật đang di chuyển sang trái thì vùng tọa độ va chạm bên trái của nhân vật sẽ được đặt bằng với tọa độ vùng va chạm bên phải của sprite, việc này giúp nhân vật không thể đi xuyên qua sprite từ bên phải sau đó gán *player.on\_left = True* cho biết nhân vật đang va chạm với sprite từ bên trái và ngược lại khi nếu có va chạm và nhân vật đang di chuyển sang phải.

## 2.9. Phương thức *vertical\_movement\_collision*

```

1 def vertical_movement_collision(self):
2     player = self.player.sprite
3     player.apply_gravity()
4     collidable_sprites = (self.terrain_sprites.sprites() + self.crates_sprites.sprites()
5                           + self.fg_palm_sprites.sprites() + self.terrain_moving_sprites.sprites())
6
7     for sprite in collidable_sprites:
8         if sprite.rect.colliderect(player.collision_rect):
9             self.player_on_moving_terrain()
10
11             if player.direction.y > 0:
12                 player.collision_rect.bottom = sprite.rect.top
13                 player.direction.y = 0
14                 player.on_ground = True
15             elif player.direction.y < 0:
16                 player.collision_rect.top = sprite.rect.bottom
17                 player.direction.y = 0
18                 player.on_ceiling = True
19
20     if player.on_ground and player.direction.y < 0 or player.direction.y > 1:
21         player.on_ground = False

```

Phương thức ***vertical\_movement\_collision*** dùng để xác định va chạm của nhân vật với các vật thể theo chiều dọc.

Gọi phương thức **`player.apply_gravity()`** để áp dụng trọng lực cho nhân vật, rồi tiếp theo cũng tạo một danh sách các sprite nhân vật có thể va chạm.

Tiếp theo phương thức sẽ lặp qua từng sprite trong danh sách và kiểm tra có va chạm giữa sprite và nhân không, nếu có ta sẽ gọi phương thức

**`self.player_on_moving_terrain()`** để xem nhân vật có đang đứng trên đối tượng ***Moving\_terrain*** không và tiếp theo xác định điều kiện tại thời điểm đó nhân vật đang di chuyển hướng xuống dưới ta đặt **`player.collission_rect.bottom`** bằng vị trí trên cùng của sprite để ngăn nhân vật bị rơi xuyên qua sprite và đứng được bình thường trên sprite rồi cài lại trọng lực về 0 và **`player.on_ground = True`** để biểu thị rằng nhân vật đang đứng trên bề mặt của địa hình. Ngược lại đối với nhân vật đang di chuyển lên trên thì sẽ đặt **`player.collission_rect.top`** bằng với vị trí dưới cùng của sprite ngăn người chơi tiếp tục đi lên và đặt **`on_ceiling`** thành True để biểu thị rằng người chơi đang chạm vào phía dưới của địa hình nào đó.

Cuối cùng ta xét **`player.on_ground`** có trả về true hay không và có đang đi lên hoặc đi xuống hay không nó sẽ cài lại **`player.on_ground`** về False để nhân vật áp dụng lại trọng lực và bị rơi xuống bình thường.

## 2.10. Phương thức `player_on_moving_terrain`

```
1 def player_on_moving_terrain(self):
2     player = self.player.sprite
3     sprites = self.terrain_moving_sprites.sprites()
4
5     for sprite in sprites :
6         if player.collission_rect.colliderect(sprite.rect):
7             player.collission_rect.topleft += pygame.math.Vector2(sprite.direction * sprite.speed * -1)
```

Phương thức trên giúp cho nhân vật khi đứng trên đối tượng ***Moving\_terrain*** sẽ di chuyển theo đối tượng và không bị đứng yên tại chỗ khi đối tượng di chuyển.

Đầu tiên phương thức cho biến **`player`** tham chiếu tới **`self.player.sprite`** và **`sprites`** tới **`self.terrain_moving_sprites.sprites()`** sau đó phương thức lặp qua các sprite trong **`sprites`** để kiểm tra sự va chạm giữa **`sprite`** và **`player`**, nếu xảy ra va chạm ra có nghĩa là nhân vật đang đứng trên đối tượng ta sẽ làm do **`player.collission_rect.topleft`** di chuyển cùng hướng và vận tốc với sprite để tạo ra hiệu ứng nhân vật vừa đứng yên trên sprite và tránh hiện tượng bị rơi khỏi sprite.



*pygame.math.Vector2(sprite.direction \* sprite.speed \* -1)* ở đoạn code này cần nhân thêm với -1 là do khi tiếp xúc với nhau thì nhân vật và sprite đang ngược hướng với nhau nên cần nhân thêm với -1 để hai đối tượng cùng chiều với nhau.

## 2.11. Phương thức `check_cannonball_collissions`

```
1 def check_cannonball_collissions(self):
2     for cannonball in self.cannonball_sprites.sprites():
3         # Xử lý va chạm với constraints_bomb_sprite
4         if pygame.sprite.spritecollide(cannonball, self.constraints_sprites, False):
5             cannonball.kill()
```

Phương thức trên kiểm tra va chạm giữa đạn pháo *cannonball* và *constraint* để làm biến mất đạn pháo khi va chạm, vòng lặp của phương thức sẽ duyệt qua các *cannonball* thuộc *self.cannonball\_sprites.sprites()* để kiểm tra va chạm với *self.constraints\_sprites*, nếu va chạm gọi *cannonball.kill()* để xóa viên đạn.

## 2.12. Phương thức `scroll_x`

```
1 def scroll_x(self):
2     player = self.player.sprite
3     player_x = player.rect.centerx
4     direction_x = player.direction.x
5
6     if player_x < screen_width / 1 and direction_x < 0:
7         self.world_shift = 8
8         player.speed = 0
9     elif player_x > screen_width - (screen_width / 1) and direction_x > 0:
10        self.world_shift = -8
11        player.speed = 0
12    else:
13        self.world_shift = 0
14        player.speed = 8
```

Phương thức *scroll\_x* được dùng để cuộn màn hình theo trục x dựa trên vị trí hiện tại của nhân vật.

Đầu tiên tạo biến *player* tham chiếu tới sprite nhân vật *self.player.sprite*, đồng tiếp theo lấy vị trí trung tâm của nhân vật và gán cho *player\_x*, *direction\_x = player.direction.x* gán hướng hiện tại của nhân vật theo trục x cho *direction\_x*.

Kiểm tra điều kiện:

*if player\_x < screen\_width / 1 and direction\_x < 0* và

*elif player\_x > screen\_width - (screen\_width / 1) and direction\_x > 0*

Hai điều kiện trên kiểm tra xem nếu vị trí và hướng di chuyển hiện tại của nhân vật có gần với mép màn hình theo trục x 1 khoảng *screen\_width / 1* về phía bên trái hoặc một

khoảng  $screen\_width - (screen\_width / 1)$  về phía bên phải. Khi đó ta sẽ làm cho tốc độ của nhân vật bằng không và `self.world_shift = 8` để khiến cho nhân vật trông như đang di chuyển thật ra là đang đứng yên con thứ di chuyển chỉ có màn hình tự cuộn theo hướng tương ứng. Ngược lại nếu không có điều kiện nào được đáp ứng, tức là người chơi không gần mép của màn hình, phương thức sẽ đặt `world_shift` về 0, vô hiệu hóa việc cuộn màn hình và cài lại tốc độ của nhân vật về lại 8 để nhân vật di chuyển lại bình thường.

### 2.13. Phương thức `get_player_on_ground`

```
1 def get_player_on_ground(self):
2     if self.player.sprite.on_ground:
3         self.player_on_ground = True
4     else:
5         self.player_on_ground = False
```

Phương thức này kiểm tra xem vị trí hiện có phải đang ở trên mặt đất hay không. Nếu có cho `self.player_on_ground = True` ngược lại `self.player_on_ground = False`.

### 2.14. Phương thức `create_landing_dust`

```
1 def create_landing_dust(self):
2     if not self.player_on_ground and self.player.sprite.on_ground and not self.dust_sprite.sprites():
3         if self.player.sprite.facing_right:
4             offset = pygame.math.Vector2(10,15)
5         else:
6             offset = pygame.math.Vector2(-10,15)
7         fall_dust_particle = ParticleEffect(self.player.sprite.rect.midbottom - offset, 'land')
8         self.dust_sprite.add(fall_dust_particle)
```

Phương thức này sẽ tạo ra hiệu ứng bụi khi nhân vật đáp xuống mặt đất sau khi nhảy lên.

Đầu tiên, phương thức kiểm tra xem người chơi có đang ở trên mặt đất không *not self.player\_on\_ground* và có đang tiếp xúc với mặt đất không *self.player.sprite.on\_ground*, và đảm bảo rằng không có bụi nào được tạo ra trước đó *not self.dust\_sprite.sprites()*. Xác định vị trí xuất phát của hiệu ứng bụi dựa trên vị trí dưới cùng và giữa của người chơi *self.player.sprite.rect.midbottom* cộng với một *offset*, để điều chỉnh vị trí hiệu ứng sao cho phù hợp với người chơi. Offset sẽ được thiết lập dựa vào hướng mà người chơi đang nhìn *self.player.sprite.facing\_right* và gán nó cho *fall\_dust\_particle*. Cuối cùng thêm nó vào *dust\_sprite* để hiển thị lên màn hình.

### 2.15. Phương thức `check_death`



```

1 def check_death(self):
2     if self.player.sprite.rect.top > screen_height:
3         self.create_overworld(self.current_level,0)

```

**Check\_death** dùng để xác định xem nếu vị trí trên cùng của nhân vật có tọa độ lớn hơn độ cao của màn hình ta sẽ xác nhận nhân vật chết và chuyển nhân vật ra lại màn hình over\_world: *self.create\_overworld(self.current\_level,0)*.

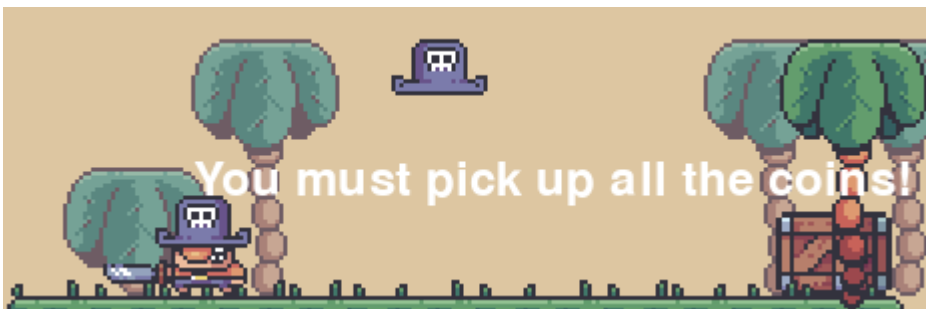
## 2.16. Phương thức check\_win

```

1 def check_win(self):
2     if pygame.sprite.spritecollide(self.player.sprite, self.goal, False) and len(self.coins_sprites)==0:
3         self.create_overworld(self.current_level, self.new_max_level)
4         self.show_message = False
5     elif pygame.sprite.spritecollide(self.player.sprite, self.goal, False) and len(self.coins_sprites):
6         self.show_message = True
7         self.message_start_time = pygame.time.get_ticks()
8         self.message_duration = 5000

```

**check\_win** sẽ kiểm tra nhân vật đã chạm tới vị trí đích và đã ăn hết số xu nếu điều kiện này đúng ta sẽ đưa nhân vật ra màn hình over\_world và mở khóa level kế tiếp và cho thuộc tính xác định việc hiển thị đoạn message *self.show\_message = False*. Ngược lại nếu chưa ăn hết xu sẽ hiện lên đoạn message nhắc người chơi phải ăn hết xu để qua màn :



## 2.17. Phương thức show\_message

```

1 def show_message(self, message, position):
2     font = pygame.font.Font(None, 36)
3     text = font.render(message, True, (255, 255, 255))
4     text_rect = text.get_rect()
5     text_rect.topright = position
6     self.display_surface.blit(text, text_rect)

```

Phương thức này chỉ đơn giản dùng để thị ra đoạn message *You must pick up all the coins!*

## 2.18. Phương thức `check_coins_collisions`

```
1 def check_coins_collisions(self):
2     collided_coins = pygame.sprite.spritecollide(self.player.sprite, self.coins_sprites, True)
3     if collided_coins:
4         self.coins_sound.play()
5         for coin in collided_coins:
6             self.change_coins(coin.value)
```

*check\_coins\_collisions* kiểm tra va chạm giữa nhân vật và xu nếu có va chạm sẽ xóa sprite của xu đi rồi phát âm thanh thể hiện cho va chạm sau đó tùy vào loại xu va chạm sẽ lấy giá trị của xu đó để tăng điểm số khi ăn xu thông qua phương thức *self.change\_coins*.

## 2.19. Phương thức `check_enemies_collisions`

```
1 def check_enemies_collisions(self):
2     enemies_collisions = (pygame.sprite.spritecollide(self.player.sprite, self.enemies_sprites, False)
3                           + pygame.sprite.spritecollide(self.player.sprite, self.canonball_sprites, False))
4     if enemies_collisions:
5         for enemy in enemies_collisions:
6             enemy_center = enemy.rect.centery
7             enemy_top = enemy.rect.top
8             player_bottom = self.player.sprite.rect.bottom
9             if enemy_top < player_bottom < enemy_center and self.player.sprite.direction.y >= 0:
10                 self.stomp_sound.play()
11                 self.player.sprite.direction.y = -13
12                 explosion_sprite = ParticleEffect(enemy.rect.center, 'explosion')
13                 self.explosion_sprites.add(explosion_sprite)
14                 enemy.kill()
15             else:
16                 self.player.sprite.get_damage()
```

Đầu tiên tạo một danh sách các sprite mà người chơi va chạm:

```
enemies_collisions =  
(pygame.sprite.spritecollide(self.player.sprite, self.enemies_sprites, False)  
+ pygame.sprite.spritecollide(self.player.sprite, self.canonball_sprites, False))
```

Tiếp theo kiểm tra điều kiện nếu va chạm xảy ra :

Tạo 3 biến *enemy\_center = enemy.rect.centery*

*enemy\_top = enemy.rect.top*

*player\_bottom = self.player.sprite.rect.bottom*

các biến trên dùng để xác định xem nhân vật sẽ va chạm với quái (enemy) ở phía trên hay không.

Sau đó xét điều kiện:

*if enemy\_top < player\_bottom < enemy\_center and self.player.sprite.direction.y >= 0:*

nếu vị trí dưới cùng của nhân vật và chạm với quái trong khoảng từ vị trí trên cùng của nhân vật tới vị trí trung tâm của nhân vật và hướng di chuyển của nhân vật đang đi xuống dưới thì ta sẽ phát âm thanh biểu thị cho việc tiêu diệt con quái ***self.stomp\_sound.play()***

và dùng dòng ***self.player.sprite.direction.y = -13*** để tạo hiệu ứng nhân vật nhảy lên 1 đoạn khi tiêu diệt quái, sau đó thêm hiệu ứng nổ khi tiêu diệt quái vào để hiển thị lên màn hình ***self.explosion\_sprites.add(explosion\_sprite)*** cuối cùng là enemy sẽ được loại bỏ ***enemy.kill()***.

Ngược lại nếu va chạm ngoài vùng đó nhân vật sẽ bị chịu sát thương từ quái qua phương thức ***self.player.sprite.get\_damage()***, nhân vật sẽ bị trừ 10 máu sau mỗi 0,5 giây nếu va chạm liên tục.

## 2.20. Phương thức run

```
1  def run(self):
2      #run the entire game / level
3
4      #sky
5      self.sky.draw(self.display_surface)
6      self.clouds.draw(self.display_surface, self.world_shift)
7
8      #bg_palm
9      self.bg_palm_sprites.update(self.world_shift)
10     self.bg_palm_sprites.draw(self.display_surface)
11
12     #terrain
13     self.terrain_sprites.update(self.world_shift)
14     self.terrain_sprites.draw(self.display_surface)
15
16     #terrain moving
17     self.terrain_moving_sprites.update(self.world_shift)
18     self.terrain_moving_collision_reverse()
19     self.terrain_moving_sprites.draw(self.display_surface)
20
21     #enemy
22     self.enemies_sprites.update(self.world_shift)
23     self.constraints_sprites.update(self.world_shift)
24     self.enemy_collision_reverse()
25     self.enemies_sprites.draw(self.display_surface)
26     self.explosion_sprites.update(self.world_shift)
27     self.explosion_sprites.draw(self.display_surface)
```

```

1  #crate
2  self.crates_sprites.update(self.world_shift)
3  self.crates_sprites.draw(self.display_surface)
4
5  #grass
6  self.grass_sprites.update(self.world_shift)
7  self.grass_sprites.draw(self.display_surface)
8
9  #coins
10 self.coins_sprites.update(self.world_shift)
11 self.coins_sprites.draw(self.display_surface)
12
13 #fg_palm
14 self.fg_palm_sprites.update(self.world_shift)
15 self.fg_palm_sprites.draw(self.display_surface)
16
17 #dust paticles
18 self.dust_sprite.update(self.world_shift)
19 self.dust_sprite.draw(self.display_surface)
20
21 #canon
22 if self.canon_file:
23     self.canon_sprites.update(self.world_shift)
24     self.canon_sprites.draw(self.display_surface)
25
26     self.canonball_sprites.update(self.world_shift, current_time=pygame.time.get_ticks())
27     self.canonball_sprites.draw(self.display_surface)
28
29     for canon in self.canon_sprites.sprites():
30         canon.fire(self.canonball_sprites, 5)
31     self.check_canonball_collissions()
32

```

```

1  #player sprtie
2  self.player.update()
3  self.horizontal_movement_collision()
4
5  self.get_player_on_ground()
6  self.vertical_movement_collision()
7  self.create_landing_dust()
8
9  self.scroll_x()
10 self.player.draw(self.display_surface)
11 self.goal.update(self.world_shift)
12 self.goal.draw(self.display_surface)
13
14 self.check_death()
15 self.check_win()
16
17 self.check_coins_collissions()
18 self.check_enemies_collissions()
19
20 if self.show_Message:
21     if pygame.time.get_ticks() - self.message_start_time < self.message_duration:
22         message = "You must pick up all the coins!"
23         message_position = (self.player.sprite.rect.right + 330, self.player.sprite.rect.top-20)
24         self.show_message(message, message_position)
25
26 #water
27 self.water.draw(self.display_surface, self.world_shift)

```

Phương thức này sẽ gọi lại tất cả các phương thức cần khởi chạy ở phía trên và ta gọi phương thức update cho tất cả các sprite với đối số *self.world\_shift* để cập nhập vị trí của sprite theo giá trị của world\_shift khi cuộn màn hình và cũng gọi tất cả các phương thức *draw* cho tất cả sprite để hiển thị các sprite lên màn hình theo đúng thứ tự và hợp lý nhất

## C.adventure\_game

Nó là module chính chứa vòng lặp để khởi tạo trò chơi.

### 1.import các thư viện và module cần thiết để khởi chạy trò chơi

```
from settings import*
from level import Level
from overworld import Overworld
from ui import UI
import pygame, sys
```

## 2.Lớp Game

### 2.1.Phương thức \_\_init\_\_

```
1 def __init__(self):
2     #game attributes
3     self.max_level = 0
4     self.max_health = 100
5     self.cur_health = 100
6     self.coins = 0
7
8     #overworld creation
9     self.overworld = Overworld(0,self.max_level,screen,self.create_level)
10    self.status = 'overworld'
11
12    #user interface
13    self.ui = UI(screen)
```

Trong phương thức \_\_init\_\_, các biến và đối tượng được khởi tạo để quản lý trạng thái và các thành phần cho trò chơi.Thực hiện khai báo các thuộc tính sau:

**max\_level:** lưu trữ level cao nhất mà người chơi đã đạt được trong trò chơi.

**max\_health:** thanh máu tối đa của người chơi gán bằng 100.

**cur\_health:** thanh máu hiện tại của người chơi.

**coins:** số xu mà người chơi đã thu thập.

**overworld**: Đối tượng Overworld được tạo ra với các tham số: cấp độ hiện tại bằng 0, cấp độ tối đa, bề mặt hiển thị (screen) và một hàm callback để tạo màn chơi level.

**status**: Biến lưu trữ trạng thái hiện tại của trò chơi, có thể là 'overworld' hoặc 'level'.

**ui**: Đối tượng UI được tạo ra với bề mặt hiển thị (screen) để hiển thị thông tin người chơi như sức khỏe và số xu.

## 2.2. Phương thức create\_level

```
1 def create_level(self, current_level):
2     self.level = Level(current_level, screen, self.create_overworld, self.change_coins, self.change_health)
3     self.status = 'level'
```

Phương thức dùng để tạo các thuộc tính dùng cho việc khởi chạy level và thuộc tính cho biết đang ở trong level, **create\_level** được gọi khi người chơi tiến tới màn mới hoặc là bắt đầu với level đầu tiên.

## 2.3. Phương thức create\_overworld

```
1 def create_overworld(self, current_level, new_max_level):
2     if new_max_level > self.max_level:
3         self.max_level = new_max_level
4     self.overworld = Overworld(current_level, self.max_level, screen, self.create_level)
5     self.status = 'overworld'
```

**create\_overworld** dùng để hiển thị ra chế độ overworld chứa các node đại diện cho các màn chơi.

Đầu tiên phương thức sẽ xét xem level mới có lớn hơn level hiện tại hay không

**if new\_max\_level > self.max\_level** nếu có sẽ cập nhật lại giá trị cho **self.max\_level** thành **new\_max\_level**.

**self.overworld = Overworld(current\_level, self.max\_level, screen, self.create\_level)**

Dòng trên tạo thuộc tính thể hiện của lớp Overworld.

Cuối cùng gán **self.status = 'overworld'** để đánh dấu đang ở trong overworld.

## 2.4. Phương thức change\_coins và change\_heath

```
1 def change_coins(self, amount):
2     self.coins += amount
3
4 def change_health(self, amout):
5     self.cur_health += amout
```

Hai phương thức trên lần lượt dùng để tính điểm cho đồng xu nhân vật và tính toán số máu hiện tại của nhân vật sau khi chịu sát thương từ enemy hay canonball.

## 2.5. Phương thức check\_game\_over

```
1 def check_game_over(self):
2     if self.cur_health <= 0 :
3         self.cur_health = 100
4         self.coins = 0
5         self.max_level = 0
6         self.overworld = Overworld(0, self.max_level, screen, self.create_level)
7         self.status = 'overworld'
```

Phương thức này được dùng để cho người chơi biết rằng mình thất bại lại level đó. Phương thức kiểm tra máu của người chơi còn đủ để tiếp tục không, nếu không đủ có nghĩa thành máu đã về 0 thì ta sẽ cài lại giá trị cho thanh máu về ban đầu là 100, trả số xu về lại 0 và cho người chơi quay trở về level 0. Tiếp tục tạo lại thuộc tính thể hiện cho Overworld với level hiện tại là 0 và cuối cùng gán *self.status* = 'overworld' để xác định đang ở trong màn hình hiển thị overworld.

## 2.6. Phương thức run

```
1 def run(self):
2     if self.status == 'overworld':
3         self.overworld.run()
4     else:
5         self.level.run()
6         self.ui.show_health(self.cur_health, self.max_health )
7         self.ui.show_coins(self.coins)
8         self.check_game_over()
```

Phương thức sử dụng thuộc tính *self.status* để xác định xem phải hiển thị màn hình nào overworld hay level. Nếu *self.status* == 'overworld' thì sẽ khởi chạy overworld và ngược lại sẽ khởi chạy level.

### 3.Vòng lặp chính khởi tạo trò chơi

```
1  #PYGAME SETUP
2  pygame.init()
3  screen = pygame.display.set_mode((screen_width,screen_height))
4  FPS = 60
5  clock = pygame.time.Clock()
6  game = Game()
7  run = True
8  while run:
9      for event in pygame.event.get():
10         if event.type == pygame.QUIT:
11             pygame.quit
12             sys.exit
13             run = False
14         game.run()
15         pygame.display.update()
16         clock.tick(FPS)
```

#### Khởi tạo màn hình game

Gọi hàm `init()` để sử dụng các hàm của `pygame`

Để tạo cửa sổ game, ta dùng `pygame.display.set_mode()` và truyền vào một tuple là

chiều rộng và dài của cửa sổ game, nó trả về dạng surface, dùng biến ***screen*** để lưu lại.

#### Kiểm soát FPS của trò chơi

Lần lượt tạo các biến FPS, clock

- ***FPS*** : giá trị là 60, là FPS của trò chơi

- ***clock***: tạo một đối tượng đồng hồ có thể được sử dụng để theo dõi thời gian

#### Tạo vòng lặp cho trò chơi

Để khởi tạo tất cả các thuộc tính cần thiết và chuẩn bị trò chơi cho việc chạy ta

tạo một đối tượng từ lớp ***Game()*** và gán nó cho ***game***

Vòng lặp `while` của game, điều kiện là `True` nghĩa là vòng lặp không có điều kiện

dừng. Trong vòng lặp `while`, tạo một vòng lặp `for` lặp qua từng sự kiện trong hàng đợi của `pygame`, nếu xuất hiện sự kiện người dùng đóng cửa sổ game thì game sẽ kết thúc, nếu không ta sẽ khởi chạy đối tượng ***game*** cho trò chơi để chạy tất cả các



hoạt động của trò chơi diễn ra, bao gồm xử lý sự kiện, cập nhật trạng thái của trò chơi và vẽ ra màn hình và ***pygame.display.update()*** để hiển thị các thay đổi mới nhất được thực hiện trong vòng lặp chính lên màn hình.

