ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

NHÓM 09

XÂY DỰNG GAME PIXEL WAR
SỬ DỤNG THƯ VIỆN PYGAME

BÁO CÁO ĐỒ ÁN MÔN HỌC

NGÔN NGỮ LẬP TRÌNH PYTHON

NGÀNH: CÔNG NGHỆ THÔNG TIN

Thành phố Hồ Chí Minh, tháng 5 năm 2024

ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

NHÓM 09

XÂY DỰNG GAME PIXEL WAR
SỬ DỤNG THƯ VIỆN PYGAME

BÁO CÁO ĐỒ ÁN MÔN HỌC

NGÔN NGỮ LẬP TRÌNH PYTHON

NGÀNH: CÔNG NGHỆ THÔNG TIN

Giảng viên phụ trách

TS. TRỊNH TẤN ĐẠT

Thành phố Hồ Chí Minh, tháng 5 năm 2024

Thành viên nhóm

STT	MSSV	Họ tên	Khối lượng(%)
1	3120410428	Triệu Khánh Quang	25%
2	3120410429	Võ Đăng Quang	25%
3	3120410437	Nguyễn Văn Tấn Quân	25%
4	3120410438	Phạm Minh Quân	25%

Lời cam đoan

Em tên là Võ Đăng Quang đại diện nhóm 09, em xin cam đoan rằng đồ án "*Xây dựng game Pixel War sử dụng thư viện Pygame*" là công trình nghiên cứu của nhóm dưới sự hướng dẫn của TS. Trịnh Tấn Đạt.

Mọi trích dẫn sử dụng trong báo cáo đều được ghi rõ nguồn tài liệu tham khảo theo đúng quy định.

Em xin hoàn toàn chịu trách nhiệm và chịu mọi hình thức kỷ luật theo quy định nếu có bất kì hành vi vi phạm, gian trá nào.

Thành phố Hồ Chí Minh, tháng 4 năm 2024

Nhóm 09

Lời cảm ơn

Trước hết, nhóm 09 chúng em xin giành lời cảm ơn đến quý thầy cô Trường Đại học Sài Gòn – khoa Công nghệ thông tin đã truyền đạt cho em những kiến thức vô cùng quý báu và bổ ích trong suốt quá trình nghiên cứu và học tập tại trường.

Tiếp đó, nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến cô TS. Trịnh Tấn Đạt, người trực tiếp hướng dẫn và tận tình chỉ bảo cho tới khi nhóm hoàn thành đồ án.

Cuối cùng, cảm ơn những người bạn trong nhóm đã luôn hỗ trợ những lúc cần thiết, giúp nhau hoàn thành đồ án này.

Mục lục

Danh mục hình ảnh

Danh mục bảng biểu

Lời mở đầu

Python là một ngôn ngữ có cấu trúc rõ ràng, thuận tiện cho người mới bắt đầu học lập trình. Đây là một ngôn ngữ lập trình đơn giản nhưng lại rất hiệu quả. Bên cạnh đó, Python có tính hướng đối tượng cao và được sử dụng rất nhiều cho các bài toán xử lý với dữ liệu lớn. Đây cũng là một ngôn ngữ tuyệt vời để tạo những nguyên mẫu (bản chạy thử – prototype). Ví dụ, bạn có thể sử dụng Pygame (thư viện viết game) để tạo nguyên mẫu game trước. Nếu thích nguyên mẫu đó có thể dùng C/C++ hay Java để viết game thật sự.

Trong bối cảnh công nghệ ngày càng phát triển như hiện nay thì nhiều thể loại game được nhiều ra đời ngày càng nhiều và thu hút được nhiều bạn trẻ. Có thể nói game đã và đang trở thành một nhu cầu giải trí thiết yếu trong cuộc sống của con người hiện nay. Một trong những thể loại game kinh điển mà khi nhắc tới không còn xa lạ gì với chúng ta, Role-Playing Game (RPG). Đây là thể loại game phổ biến, nơi người chơi điều khiển một nhân vật trong thế giới ảo, trải nghiệm các cuộc phiêu lưu, chiến đấu với kẻ thù và hoàn thành nhiệm vụ. RPG thu hút người chơi bởi sự đa dạng, khả năng sáng tạo và tính giải trí cao.

Sử dụng các kiến thức học được, nhóm chúng em chọn “Xây dựng game Pixel War sử dụng thư viện Pygame” để làm đồ án cho môn học này. Quá trình nghiên cứu nhóm em sẽ trình bày sơ lược về ngôn ngữ lập trình Python cũng như cấu trúc của nó. Kết hợp việc quan sát và thực hành để tạo ra một phiên bản RPG trực quan và đơn giản nhất cho đồ án của mình.

Chương 1. SƠ LƯỢC VỀ ĐỀ TÀI

1.1. Tổng quan Python

1.1.1. Lịch sử phát triển

Vào đầu những năm 1980, ABC là một ngôn ngữ lập trình bậc cao, được phát triển bởi Leo Geurts và Lambert Meertens tại Đại học Amsterdam. ABC được thiết kế để trở thành một ngôn ngữ đơn giản, dễ học và dễ sử dụng, phù hợp cho việc giảng dạy lập trình. Đặc điểm của ABC là cú pháp đơn giản, dễ đọc, kiểu dữ liệu động, hỗ trợ lập trình cấu trúc và lập trình hướng đối tượng. ABC được sử dụng chủ yếu cho việc giảng dạy lập trình. ABC cũng có thể được sử dụng để phát triển các ứng dụng đơn giản.

Với mục tiêu đơn giản hoá hơn các cú pháp và tăng cường khả năng xử lý, Guido Van Rossum bắt đầu phát triển Python. Công cuộc thiết kế bắt đầu vào cuối những năm 1980 và vào tháng 2 năm 1991, Python được phát hành lần đầu tiên. Sau hơn 30 năm phát triển và cải tiến, Python đã trở thành một trong những ngôn ngữ lập trình phổ biến nhất hiện nay.

HÌNH 1.1. NGÔN NGỮ PYTHON

Python được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, bao gồm: Phát triển web, Khoa học dữ liệu, Học máy, Tự động hóa, Phát triển phần mềm,… Cộng đồng lớn và hoạt động tích cực cũng là một trong những lí do làm nên sự thành công của Python. Trong hơn 30 năm phát triển, Python có một thư viện tiêu chuẩn phong phú với nhiều chức năng sẵn có và được cấp nhật thường xuyên các tính năng mới như: numpy, scipy, matplotlib , sklearn,…

1.1.2. So sánh Python với các ngôn ngữ lập trình khác

Python, Java và C++ là ba ngôn ngữ lập trình phổ biến nhất hiện nay. Mỗi ngôn ngữ đều có những ưu điểm và nhược điểm riêng, phù hợp với những mục đích sử dụng khác nhau.

Dưới đây là bảng so sánh chi tiết giữa Python, Java và C++:

Tiêu chí	Python	Java	C++
Cú pháp	Dễ đọc, dễ học	Dễ đọc, dễ học	Khó đọc, khó học
Kiểu dữ liệu	Động	Tĩnh	Tĩnh
Quản lý bộ nhớ	Tự động	Tự động	Thủ công
Hiệu suất	Chậm	Nhanh	Nhanh
Khả năng mở rộng	Cao	Cao	Thấp
Thư viện tiêu chuẩn	Phong phú	Phong phú	Ít phong phú
Ứng dụng	Phát triển web, khoa học dữ liệu, học máy, tự động hóa	Phát triển web, ứng dụng di động, phần mềm doanh nghiệp	Hệ thống nhúng, trò chơi, phần mềm đồ họa

BẢNG 1. 1. BẢNG SO SÁNH PYTHON VỚI CÁC NGÔN NGỮ KHÁC

Qua bản so sánh trên, ta có thể kết luận được rằng:

- Python là lựa chọn tốt nhất dành cho người mới học lập trình vì Python có cú pháp đơn giản và dễ học, giúp bạn dễ dàng tiếp cận với lập trình.
- Nếu muốn phát triển web, Java hoặc Python mới là sự lựa chọn tốt. Java và Python đều có các framework web mạnh mẽ và cộng đồng hỗ trợ lớn.
- Nếu muốn nghiên cứu khoa học dữ liệu hoặc training AI, thì Python là sự lựa chọn tối ưu vì cú pháp và các thư viện hỗ trợ rất tốt cho mảng này.

1.2. Giới thiệu về thư viện Pygame

Pygame là một thư viện mã nguồn mở của Python được sử dụng để phát triển game. Nó cung cấp bộ công cụ cần thiết để tạo ra các thành phần cơ bản đến nâng cao. Kể cả khi là người mới cũng có thể dễ dàng sử dụng thư viện này.

1.2.1. Tính năng của Pygame

- Đồ họa: Pygame cung cấp các chức năng để vẽ hình ảnh, sprite và các hiệu ứng đồ họa khác.
- Âm thanh: Pygame cung cấp các chức năng để phát nhạc và âm thanh.
- Đầu vào: Pygame cung cấp các chức năng để xử lý các sự kiện đầu vào như nhấp chuột, di chuyển chuột và nhấn phím.
- Khả năng tương thích: Pygame có thể được sử dụng trên nhiều hệ điều hành khác nhau, bao gồm Windows, Mac và Linux.

1.2.2. Ưu điểm của Pygame

- Dễ học và sử dụng: Pygame có cú pháp đơn giản và dễ hiểu, giúp cho người mới bắt đầu dễ dàng tiếp cận.
- Linh hoạt: Pygame có thể được sử dụng để tạo ra nhiều loại trò chơi khác nhau, từ các trò chơi đơn giản đến các trò chơi phức tạp.
- Mã nguồn mở: Pygame là mã nguồn mở, nghĩa là nó miễn phí và có thể được sử dụng bởi bất kỳ ai.

- Cộng đồng lớn: Pygame có một cộng đồng lớn và hoạt động tích cực, luôn sẵn sàng hỗ trợ lẫn nhau.

1.2.3. Nhược điểm của Pygame

- Khó khăn trong việc tối ưu hóa hiệu suất cho các game 3D phức tạp: Mặc dù Pygame có hiệu suất cao, nhưng việc tối ưu hóa hiệu suất cho các game 3D phức tạp có thể đòi hỏi nhiều kỹ năng và kinh nghiệm hơn.
- Hạn chế về hỗ trợ đồ họa: Pygame không hỗ trợ tất cả các tính năng đồ họa tiên tiến nhất, đặc biệt là trong lĩnh vực game 3D.
- Tốc độ phát triển chậm: Pygame là một dự án mã nguồn mở, do đó tốc độ phát triển của thư viện có thể chậm hơn so với các thư viện thương mại.
- Ít tài liệu hướng dẫn cho các tính năng nâng cao: Một số tính năng nâng cao của Pygame có thể không có nhiều tài liệu hướng dẫn và ví dụ minh họa chi tiết.

Chương 2. XÂY DỰNG TRÒ CHƠI

2.1. Mô tả trò chơi

2.1.1. Bối cảnh trò chơi

Pixel War là một tựa game lấy giả tưởng lấy bối cảnh 1 chiến binh Viking tên John sống sót sau vụ đắm thuyền, trôi dạt lên hòn đảo bí ẩn đầy rẫy quái vật. Nhân vật của chúng ta với khả năng thông thạo tất cả các loại vũ khí và có thể sử dụng phép thuật một cách mạnh mẽ. Nhiệm vụ của trò chơi là điều khiển John tiêu diệt tất cả các quái vật trên bản đồ để sống sót trên hòn đảo này.

2.1.2. Thể loại trò chơi

Indie, sinh tồn, chơi đơn, hành động, độ khó cao, pixel, có tính chơi lại, góc nhìn thử ba, chơi miễn phí, nhập vai phiêu lưu, đơn giản.

2.2. Xây dựng trò chơi

2.2.1. Thiết lập khung nhìn

Để sử dụng trò chơi ta cần khai báo 2 thư viện: pygame và sys ở file main.py.

```
import pygame, sys
from settings import *
```

HÌNH 2.1. THÊM THƯ VIỆNC CHÍNH CHO FILE MAIN.PY

- pygame: Thư viện đồ hoạ và âm thanh cho game.
- sys: Thư viện builtin của Python, nó chứa các thông tin liên quan đến chính chương trình python interpreter bạn đang chạy.

Tiếp theo tạo file settings.py lưu tất cả các cấu hình của game vào. Sau đó import settings vào trong main.

```
from level import Level
```

HÌNH 2.2. THÊM FILE LEVEL.PY

Trong file settings.py ta có các cài đặt sau:

```
WIDTH    = 1280
HEIGHT   = 720
FPS      = 60
TILESIZE = 64
```

HÌNH 2.3. CÁC SETTINGS CHÍNH TRONG TRÒ CHƠI

- WIDTH và HEIGHT: là thông số xác định kích thước cửa sổ trò chơi. Ví dụ, chiều rộng của cửa sổ là 1280 pixel và chiều cao là 720 pixel.
- FPS: là thông số thể hiện số lượng hình ảnh được hiển thị trên màn hình trò chơi mỗi giây, FPS càng cao thì game hiển thị càng mượt. Trong ví dụ này, trò chơi sẽ được vẽ lại 60 lần mỗi giây.
- TILESIZE: là kích thước của nền trò chơi

2.2.2. Thiết lập môi trường

```
# general setup
pygame.init()
self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('PixelWar Project')
self.clock = pygame.time.Clock()

self.level = Level()

# sound
main_sound = pygame.mixer.Sound('../audio/main.mp3')
main_sound.set_volume(0.5)
main_sound.play(loops = -1)
```

HÌNH 2.4. THIẾT LẬP MÔI TRƯỜNG

- pygame.init(): dòng này khởi tạo thư viện Pygame, thư viện cần thiết để tạo trò chơi trong Python. Hàm này phải được gọi trước khi sử dụng bất kỳ chức năng nào khác của Pygame.
- pygame.display.set_mode((WIDTH, HEIGHT)): kích thước cửa sổ của game sẽ được xác định bởi WIDTH và HEIGHT (1280 pixel x 720 pixel).
- pygame.display.set_caption('PixelWar Project'): Dòng này đặt tiêu đề cho cửa sổ trò chơi hiển thị ở đầu khung cửa sổ. Tiêu đề trong trường hợp này được đặt thành "PixelWar Project".
- self.clock = pygame.time.Clock(): Dòng này tạo một đối tượng đồng hồ bằng cách sử dụng mô-đun pygame.time. Đồng hồ này được sử dụng để quản lý tốc độ khung hình của trò chơi, đảm bảo trải nghiệm hoạt hình mượt mà.
- self.level = Level(): Tạo một lớp Level để quản lý các vấn đề liên quan đến cấp độ trong game và gán nó cho biến self.level trong lớp hiện tại.
- Các dòng này tiếp theo được sử dụng để thiết lập nhạc nền cho trò chơi:

- main_sound = pygame.mixer.Sound('../audio/main.mp3'): Dòng này tải tệp âm thanh (main.mp3) nằm trong thư mục con tên "audio" liên quan đến tập lệnh Python.
- main_sound.set_volume(0.5): Dòng này đặt âm lượng của tệp âm thanh thành 50% âm lượng của máy.
- main_sound.play(loops = -1): Dòng này để đặt tệp âm thanh lặp lại vô hạn (loops = -1).

2.2.3. Khởi tạo hitbox

Tạo 1 file tên tile.py để chứa các thiết lập liên quan đến hitbox trong game. Mỗi đối tượng (nhân vật, quái, cây cỏ, tường,…) là 1 ô vuông (tile) trong game. Để các đối tượng có thể tương tác được với nhau lớp Tile sẽ được sẽ sử dụng để quản lý chúng.

```
class Tile(pygame.sprite.Sprite):
    def __init__(self, pos, groups, sprite_type, surface = pygame.Surface((TILESIZE, TILESIZE))):
        super().__init__(groups)
        self.sprite_type = sprite_type
        y_offset = HITBOX_OFFSET[sprite_type]
        self.image = surface
        if sprite_type == 'object':
            self.rect = self.image.get_rect(topleft = (pos[0], pos[1] - TILESIZE))
        else:
            self.rect = self.image.get_rect(topleft = pos)
        self.hitbox = self.rect.inflate(0, y_offset)
```

HÌNH 2.5. KHỞI TẠO HITBOX

- def __init__(self, pos, groups, sprite_type, surface = pygame.Surface((TILESIZE, TILESIZE))): Đây là hàm khởi tạo của lớp Tile, trong đó:
  - pos: Tham số này là một tuple chứa vị trí (x, y) của ô tile trên màn hình.
  - groups: Tham số này là một hoặc nhiều nhóm sprite mà ô tile sẽ được thêm vào.
  - sprite_type: Chuỗi ký tự xác định loại của ô tile (ví dụ: 'ground', 'wall', 'object').
  - surface (tham số tuỳ chọn): Tham số này để tạo ô tile cho cửa sổ trò chơi lấy kích thước của nền trò chơi làm mặc định.
- super().__init__(groups): Hàm khởi tạo của lớp cha pygame.sprite.Sprite, truyền tham số groups để thêm ô tile vào các nhóm sprite đã cho.

Trở lại với file settings.py ta tiếp tục thêm HITBOX_OFFSET để thêm vị trí của hitbox cho các loại ô tile khác nhau:

```
HITBOX_OFFSET = {
    'player': -26,
    'object': -40,
    'grass': -10,
    'invisible': 0
}
```

HÌNH 2.6. THÊM SETTINGS CHO HITBOX

Về phía file tile.py:

- y_offset = HITBOX_OFFSET[sprite_type]: Đây là lấy giá trị lấy từ settings.py là HITBOX_OFFSET với key là sprite_type gán cho biến y_offset.

- self.image = surface: Thuộc tính này lưu trữ hình ảnh đại diện cho ô tile.

Cách các hitbox hoạt động như sau:

➢ Trường hợp sprite_type là 'object':

- self.rect = self.image.get_rect(topleft = (pos[0], pos[1] - TILESIZE)): Tạo một hình chữ nhật bao quanh image (được lưu trong self.rect) và đặt vị trí trên cùng bên trái (topleft) của hình chữ nhật tại tọa độ (pos[0], pos[1] - TILESIZE).

➢ Trường hợp khác (sprite_type không phải 'object'):

- self.rect = self.image.get_rect(topleft = pos): Tạo một hình chữ nhật bao quanh image và đặt vị trí trên cùng bên trái của hình chữ nhật tại tọa độ là pos.

Kết thúc vòng lặp

- self.hitbox = self.rect.inflate(0, y_offset): Tạo hitbox dựa trên self.rect và nới rộng vùng này theo trục Y (y_offset) để tính toán hitbox chính xác hơn. Giá trị y_offset được lấy từ HITBOX_OFFSET tùy theo loại ô tile đã khai báo ở settings.py.

2.2.4. Khởi tạo camera di chuyển theo nhân vật và background

Tạo 1 file level.py để chứa tất cả các vấn đề liên quan đến cấp độ trong trò chơi. Trong file này ta cần các thư viện cũng như các file sau để trò chơi được vận hành một cách trơn tru nhất.

```
from tile import Tile
from player import Player
from debug import debug
from support import *
from random import choice, randint
from weapon import Weapon
from ui import UI
from enemy import Enemy
from particles import AnimationPlayer
from magic import MagicPlayer
from upgrade import Upgrade
```

HÌNH 2.7. CÁC THƯ VIỆN CẦN THIẾT TRONG FILE LEVEL.PY

- tile: ta cần file tile.py đã tạo trước đó để sử dụng hitbox.
- player: tạo 1 file player.py để chứa các thiết lập liên quan đến nhân vật.
- debug: tạo 1 file debug.py để hiện các thông báo của trò chơi trên màn hình máy.
- support: tạo 1 file support.py để quản lý các file csv và hình ảnh từ các thư mục khác nhau.
- random: sử dụng thư viện random để chọn ngẫu nhiên các object (vật thể) như cỏ (grass).
- ui: tạo 1 file ui.py để quản lý các giao diện trong trò chơi.
- enemy: tạo 1 file enemy.py để chứa các thiết lập liên quan tới quái.
- particles: tạo 1 file particles.py để chứa các thiết lập liên quan đến chuyển động của nhân vật.
- magic: tạo 1 file magic.py để chứa các thiết lập phép thuật của nhân vật.
- upgrade: tạo 1 file upgrade.py để chứa các thiết lập của hệ thống nâng cấp nhân vật.

Khởi tạo hàm create_map để tạo ra bố cục và sắp xếp các yếu tố khác nhau trong trò chơi.

```
def create_map(self):
    layouts = {
        'boundary': import_csv_layout('../map/map_FloorBlocks.csv'),
        'grass': import_csv_layout('../map/map_Grass.csv'),
        'object': import_csv_layout('../map/map_Objects.csv'),
        'entities': import_csv_layout('../map/map_Entities.csv')
    }
    graphics = {
        'grass': import_folder('../graphics/grass'),
        'objects': import_folder('../graphics/objects')
    }
```

HÌNH 2.8. TẠO NỀN CHO TRÒ CHƠI

Hàm bắt đầu bằng việc xác định: layouts và graphics.

- layouts: là khóa đại diện cho các kiểu ô khác nhau như "boundary", "grass", "object" và "entities". Giá trị là kết quả của việc gọi hàm import_csv_layout , hàm này đọc dữ liệu từ các tệp CSV tương ứng.
- graphics: là khoá để lưu trữ hình ảnh của các loại cỏ (grass) và vật thể (objects) đã được lưu trước đó.

Tạo vòng lặp qua các file csv đã đọc ở trên:

```
for style, layout in layouts.items():
    for row_index, row in enumerate(layout):
        for col_index, col in enumerate(row):
            if col != '-1':
                x = col_index * TILESIZE
                y = row_index * TILESIZE
```

HÌNH 2.9. ĐỌC FILE CSV

- row_index: Chỉ mục của hàng hiện tại đang được xử lý.
- row: Danh sách thực tế đại diện cho hàng ô hiện tại.
- col_index: Chỉ mục của cột hiện tại trong hàng.
- col: Giá trị tại vị trí ô hiện tại trong bố cục.
- if col != '-1': Bỏ qua việc xử lý các ô trống được đánh dấu bằng '-1' trong csv.

Sau khi đã kiểm tra và xử lý file csv, tiếp theo ta tạo ô tile dựa trên style. Tùy thuộc vào style, các ô tile khác nhau sẽ được tạo:

```
if style == 'boundary':
    Tile((x, y), [self.obstacle_sprites], 'invisible')

if style == 'grass':
    random_grass_image = choice(graphics['grass'])
    Tile(
        (x,y),
        [self.visible_sprites, self.obstacle_sprites, self.attackable_sprites],
        'grass',
        random_grass_image
    )

if style == 'object':
    surf = graphics['objects'][int(col)]
    Tile((x, y), [self.visible_sprites, self.obstacle_sprites], 'object', surf)
```

HÌNH 2.10. XỬ LÝ DỮ LIỆU TRONG FILE CSV

- boundary: Nếu style là "boundary", một ô tile được tạo với đồ họa trống nhưng được đánh dấu là chướng ngại vật (invisible hitbox).
- grass: Nếu style là "grass", một ô tile được tạo với hình ảnh cỏ ngẫu nhiên được hiển thị trên cửa sổ.
- object: Nếu style là "object", một ô tile được tạo với bề mặt hình ảnh tương ứng được lấy từ graphics['objects'] sử dụng giá trị từ file csv và chỉ mục là [int(col)].

Tiếp theo ta sẽ hiển thị vị trí thực thể gồm nhân vật và quái vật. Trong trò chơi này khi được bắt đầu, nhân vật và quái sẽ đứng ở một vị trí cố định:

```
if style == 'entities':
    if col == '394':
        self.player = Player(
            (x,y),
            [self.visible_sprites],
            self.obstacle_sprites,
            self.create_attack,
            self.destroy_attack,
            self.create_magic
        )
    else:
        if col == '390': monster_name = 'bamboo'
        elif col == '391': monster_name = 'spirit'
        elif col == '392': monster_name ='raccoon'
        else: monster_name = 'squid'
        Enemy(
            monster_name,
            (x,y),
            [self.visible_sprites,self.attackable_sprites],
            self.obstacle_sprites,
            self.damage_player,
            self.trigger_death_particles,
            self.add_exp
        )
```

HÌNH 2.11. THIẾT LẬP VỊ TRÍ CỦA CÁC VẬT THỂ VÀ QUÁI

Nếu style là "entities" thì sẽ tiến hành kiểm tra trong file csv giá trị ô hiện tại (col).

- Nhân vật col mặc định sẽ là "394", một ô tile dành cho nhân vật được tạo bao gồm các thuộc tính sau:
  - Vị trí cố định.
  - Lớp sprite (hiển thị, chướng ngại vật)
  - Chức năng tấn công/phá hủy vật thể/sử dung phép thuật.
- Quái vật: có các col như sau: "390" là bamboo, "391" là spirit, "392" là raccoon và "393" là squid. Khi kiểm tra đúng col thì ô tile cho quái được tạo với các thuộc tính sau:

- Loại quái vật dựa trên giá trị col.
- Vị trí dựa trên chỉ mục.
- Lớp sprite hiển thị, có thể tấn công và chướng ngại vật.
- Chức năng gây sát thương cho nhân vật, kích hoạt hiệu ứng chết và thưởng điểm kinh nghiệm cho nhân vật.

Sau khi tạo xong nền cho nhân vật và quái có thể tương tác với các vật thể xung quanh thì trò chơi đã hoàn thành được nửa chặng đường. Bây giờ ta cần xây dựng một hệ thống camera di chuyển theo nhân vật nếu không thì ta không thể điều khiển nhân vật nếu nó đi khỏi tầm mắt của màn hình. Để làm được điều đó ta khởi tạo hàm YsortCameraGroup.

```
class YSortCameraGroup(pygame.sprite.Group):
```

HÌNH 2.12. KHỞI TẠO HÀM CHỨA CAMERA DI CHUYỂN THEO VỊ TRÍ NHÂN VẬT

Trong đó khởi tạo phương thức __init__ để định nghĩa các đối tượng sẽ được sử dụng.

```
def __init__(self):

    # general setup
    super().__init__()
    self.display_surface = pygame.display.get_surface()
    self.half_width = self.display_surface.get_size()[0] // 2
    self.half_height = self.display_surface.get_size()[1] // 2
    self.offset = pygame.math.Vector2()

    # creating the floor
    self.floor_surf = pygame.image.load('../graphics/tilemap/ground.png').convert()
    self.floor_rect = self.floor_surf.get_rect(topleft = (0, 0))
```

HÌNH 2.13. CÁC NỘI DUNG TRONG HÀM YSORTCAMERAGROUP

- super().__init__(): Dòng này gọi hàm khởi tạo của lớp cha (pygame.sprite.Group), đảm bảo khởi tạo đúng để quản lý các sprite.
- self.display_surface = pygame.display.get_surface(): Dòng này lấy bề mặt hiển thị chính nơi trò chơi được hiển thị.
- self.half_width = self.display_surface.get_size()[0] // 2: Dòng này tính toán một nửa chiều rộng của bề mặt hiển thị, có thể được sử dụng để căn giữa camera.

- self.half_height = self.display_surface.get_size()[1] // 2: Tương tự như trên, dòng này tính toán một nửa chiều cao của bề mặt hiển thị.
- self.offset = pygame.math.Vector2(): Dòng này tạo một đối tượng pygame.math.Vector2 để lưu trữ độ lệch camera.
- self.floor_surf = pygame.image.load('../graphics/tilemap/ground.png').convert(): Dòng này tải ảnh nền và chuyển đổi để hiển thị nhanh hơn.
- self.floor_rect = self.floor_surf.get_rect(topleft = (0, 0)): Dòng này lấy một hình chữ nhật biểu thị vị trí và kích thước của ảnh nền.

Sau khi có các định nghĩa ở trên ta khởi tạo phương thức custom_draw để tính toán độ sai lệch camera trên trục X và Y sau đó mới tiến hành vẽ hình ảnh của nhân vật lên bề mặt đã được tính toán trước:

```
def custom_draw(self, player):

    # getting the offset
    self.offset.x = player.rect.centerx - self.half_width
    self.offset.y = player.rect.centery - self.half_height

    # drawing the floor
    floor_offset_pos = self.floor_rect.topleft - self.offset
    self.display_surface.blit(self.floor_surf, floor_offset_pos)

    # for sprite in self.sprites():
    for sprite in sorted(self.sprites(), key = lambda sprite: sprite.rect.centery):
        offset_pos = sprite.rect.topleft - self.offset
        self.display_surface.blit(sprite.image, offset_pos)
```

HÌNH 2.14. PHƯƠNG THỨC CUSTOM_DRAW

- self.offset.x = player.rect.centerx - self.half_width: Dòng này tính toán độ lệch camera trên trục X bằng cách trừ một nửa chiều rộng hiển thị khỏi vị trí X trung tâm của nhân vật. Điều này làm cho camera tập trung vào nhân vật.
- self.offset.y = player.rect.centery - self.half_height: Tương tự như trên, dòng này tính toán độ lệch camera trên trục Y dựa trên vị trí Y trung tâm của nhân vật.

- floor_offset_pos = self.floor_rect.topleft - self.offset: Dòng này tính toán vị trí của ảnh nền trên màn hình bằng cách trừ độ lệch camera khỏi góc trên bên trái ban đầu của nó.
- self.display_surface.blit(self.floor_surf, floor_offset_pos): Dòng này sao chép (vẽ) ảnh nền lên bề mặt hiển thị chính tại vị trí được tính toán.
- for sprite in sorted(self.sprites(), key = lambda sprite: sprite.rect.centery): Dòng này lặp qua các sprite trong nhóm, nhưng trước tiên, chúng được sắp xếp dựa trên vị trí Y trung tâm của chúng (sử dụng hàm lambda làm khóa).
- offset_pos = sprite.rect.topleft - self.offset: Tương tự như nền, dòng này tính toán vị trí trên màn hình của mỗi sprite bằng cách trừ độ lệch camera khỏi vị trí ban đầu của nó.
- self.display_surface.blit(sprite.image, offset_pos): Dòng này sao chép từng sprite lên bề mặt hiển thị chính tại vị trí được tính toán tương ứng. Việc sắp xếp này đảm bảo rằng các sprite ở xa hơn (trung tâm Y cao hơn) được vẽ phía sau các sprite gần hơn.

Cuối cùng, tạo phương thức enemy_update để xử lý hành vi của quái với nhân vật

```
def enemy_update(self, player):
    enemy_sprites = [sprite for sprite in self.sprites() if hasattr(sprite, 'sprite_type') and sprite.sprite_type == 'enemy']
    for enemy in enemy_sprites:
        enemy.enemy_update(player)
```

HÌNH 2.15. PHƯƠNG THỨC ENEMY_UPDATE

- enemy_sprites = [sprite for sprite in self.sprites() if hasattr(sprite, 'sprite_type') and sprite.sprite_type == 'enemy']: Dòng này tạo một danh sách chỉ chứa các sprite quái. Nó lọc dựa trên việc sprite có thuộc tính sprite_type được đặt thành "enemy" hay không.
- for enemy in enemy_sprites: Vòng lặp này lặp qua các sprite của quái.
- enemy.enemy_update(player): Dòng này gọi phương thức enemy_update có thể tồn tại trên lớp sprite kẻ thù. Phương thức này có thể xử lý logic của quái và cập nhật hành vi của chúng dựa trên vị trí hoặc hành động của nhân vật.

2.3. Xây dựng nhân vật

HÌNH 2.16. NHÂN VẬT CHÍNH CỦA TRÒ CHƠI

Đây là nhân vật chính của trò chơi. Nhân vật có bốn kiểu di chuyển: Lên, Xuống, Trái, Phải. Để cho nhân vật có thể chuyển động cũng như tương tác với mọi thứ xung quanh, ta tạo file player.py để chứa các thiết lập liên quan đến nhân vật.

```
class Player(Entity):
    def __init__(self, pos, groups, obstacle_sprites, create_attack, destroy_attack, create_magic):
        super().__init__(groups)
        self.image = pygame.image.load('../graphics/test/player.png').convert_alpha()
        self.rect = self.image.get_rect(topleft = pos)
        self.hitbox = self.rect.inflate(-6,HITBOX_OFFSET['player'])
```

HÌNH 2.17. ĐỌC FILE ĐỂ NHÂN VẬT HIỂN THỊ TRÊN TRÒ CHƠI

Trong file ta khởi tạo hàm Player và vẽ cho nhân vật ô tile để hitbox của nhân vật có thể tương tác với các thực thể trên bản đồ. Tiếp đến ta thêm các hành động di chuyển cho nhân vật.

```
def import_player_assets(self):
    character_path = '../graphics/player/'
    self.animations = {
        'up': [], 'down': [], 'left': [], 'right': [],
        'right_idle':[], 'left_idle':[], 'up_idle':[], 'down_idle':[],
        'right_attack':[], 'left_attack':[], 'up_attack':[], 'down_attack':[]
    }

    for animation in self.animations.keys():
        full_path = character_path + animation
        self.animations[animation] = import_folder(full_path)
```

HÌNH 2.18. THIẾT LẬP CÁC NÚT DI CHUYỂN CHO NHÂN VẬT (1)

- character_path: Biến này lưu trữ đường dẫn đến thư mục chứa các hình ảnh của nhân vật trong thư mục graphics/player.
- Khối lệnh self.animations = {...} chứa các khoá thể hiện hướng di chuyển như: 'up', 'down', 'left', 'right' hoặc trạng thái kết hợp với hướng di chuyển như: 'right_idle', 'left_idle'.

- for animation in self.animations.keys(): vòng lặp để lặp qua từng khóa trong animations. Bên trong vòng lặp:

- full_path = character_path + animation: Biến full_path được tạo ra để lưu trữ đường dẫn đầy đủ đến thư mục chứa các hình ảnh hoạt hình cho một hướng di chuyển/trạng thái cụ thể. Ví dụ, nếu animation là 'up', thì full_path sẽ bằng '../graphics/player/up/'.

- self.animations[animation] = import_folder(full_path): Dòng lệnh này chỉ ra rằng hàm import_folder có trách nhiệm đọc các hình ảnh từ thư mục được cung cấp (full_path) và trả về một danh sách chứa các hình ảnh đó. Danh sách này sau đó được gán vào khóa tương ứng trong animations.

Sau khi nhân vật đã được hiển thị trên cửa sổ, phần code tiếp theo ta sẽ tiến hành xử lý các thao tác nhập từ bàn phím của người chơi để điều khiển nhân vật.

```
def input(self):
    if not self.attacking:
        keys = pygame.key.get_pressed()

        # movement input
        if keys[pygame.K_UP]:
            self.direction.y = -1
            self.status = 'up'
        elif keys[pygame.K_DOWN]:
            self.direction.y = 1
            self.status = 'down'
        else:
            self.direction.y = 0

        if keys[pygame.K_RIGHT]:
            self.direction.x = 1
            self.status = 'right'
        elif keys[pygame.K_LEFT]:
            self.direction.x = -1
            self.status = 'left'
        else:
            self.direction.x = 0
```

HÌNH 2.19. THIẾT LẬP CÁC NÚT DI CHUYỂN CHO NHÂN VẬT (2)

- if not self.attacking: Lệnh này sẽ kiểm tra xem nhân vật có đang tấn công hay không, nếu không, các lệnh bên trong lệnh if sẽ được thực thi.
- keys = pygame.key.get_pressed(): Dòng lệnh này lấy trạng thái của tất cả các phím trên bàn phím. Kết quả trả về là một danh sách keys, trong đó:

- Mỗi phần tử trong danh sách tương ứng với một phím trên bàn phím.
- Giá trị của phần tử là True nếu phím đó đang được bấm, False nếu không thì sẽ không có gì xảy ra.

2.4. Hệ thống vũ khí và phép thuật

2.4.1. Chuyển đổi vũ khí và phép thuật

```
if keys[pygame.K_q] and self.can_switch_weapon:
    self.can_switch_weapon = False
    self.weapon_switch_time = pygame.time.get_ticks()

    if self.weapon_index < len(list(weapon_data.keys())) - 1:
        self.weapon_index += 1
    else:
        self.weapon_index = 0

    self.weapon = list(weapon_data.keys())[self.weapon_index]

if keys[pygame.K_e] and self.can_switch_magic:
    self.can_switch_magic = False
    self.magic_switch_time = pygame.time.get_ticks()

    if self.magic_index < len(list(magic_data.keys())) - 1:
        self.magic_index += 1
    else:
        self.magic_index = 0

    self.magic = list(magic_data.keys())[self.magic_index]
```

HÌNH 2.20. CHUYỂN ĐỔI VŨ KHÍ VÀ PHÉP THUẬT

- keys: Biến lưu trữ trạng thái của các phím được bấm.
- pygame.K_q: Đây là hằng số đại diện cho phím "Q" trên bàn phím.
- self.can_switch_weapon: Biến kiểm tra xem có cho phép chuyển vũ khí hay không (khóa chuyển đổi tạm thời).
- self.weapon_switch_time: Lưu trữ thời gian bấm phím chuyển vũ khí gần nhất.
- weapon_data: Biến chứa thông tin vũ khí.
- self.weapon_index: Chỉ mục của vũ khí đang được trang bị.

- self.can_switch_magic: Biến kiểm tra xem có cho phép chuyển phép thuật hay không (khóa chuyển đổi tạm thời).
- self.magic_switch_time: Lưu trữ thời gian bấm phím chuyển phép thuật gần nhất.
- magic_data: Biến chứa thông tin về phép thuật.
- self.magic_index: Chỉ mục của phép thuật đang được trang bị.
- self.weapon: Vũ khí đang được trang bị.
- self.magic: Phép thuật đang được trang bị.

Nếu phím "Q" được bấm (keys[pygame.K_q]) và cho phép chuyển vũ khí (self.can_switch_weapon là True):

- Khóa chuyển vũ khí tạm thời (self.can_switch_weapon = False).
- Lưu trữ thời gian bấm phím chuyển vũ khí (self.weapon_switch_time = pygame.time.get_ticks()).
- Tăng chỉ mục vũ khí (self.weapon_index) nhưng kiểm tra xem có vượt quá giới hạn không (dùng len(list(weapon_data.keys())) để lấy số lượng vũ khí).
- Nếu vượt (self.weapon_index >= len(weapon_data.keys())), đặt lại về chỉ mục đầu tiên (self.weapon_index = 0).
- Cập nhật vũ khí đang trang bị (self.weapon = list(weapon_data.keys())[self.weapon_index]). Chuyển đổi dictionary weapon_data thành danh sách để lấy ra tên vũ khí theo chỉ mục.

Nếu phím “E” được bấm: Logic tương tự như chuyển vũ khí (bấm phím “Q”) nhưng sử dụng các biến liên quan đến phép thuật (self.can_switch_magic, self.magic_switch_time, magic_data, self.magic_index).

2.4.2. Hệ thống vũ khí

```
# weapons
weapon_data = {
    'sword': {'cooldown': 100, 'damage': 15,'graphic': '../graphics/weapons/sword/full.png'},
    'lance': {'cooldown': 400, 'damage': 30,'graphic': '../graphics/weapons/lance/full.png'},
    'axe': {'cooldown': 300, 'damage': 20, 'graphic': '../graphics/weapons/axe/full.png'},
    'rapier':{'cooldown': 50, 'damage': 10, 'graphic': '../graphics/weapons/rapier/full.png'},
    'sai':{'cooldown': 80, 'damage': 15, 'graphic': '../graphics/weapons/sai/full.png'}
}
```

HÌNH 2.21. DỮ LIỆU VŨ KHÍ

weapon_data là một dictionary dùng để lưu trữ thông tin về các loại vũ khí trong trò chơi, với mỗi key (khóa) là tên của một loại vũ khí (ví dụ: 'sword', 'lance',…) và value (giá trị) là một dictionary khác chứa các thuộc tính của loại vũ khí đó.

Các thuộc tính của weapon_data bao gồm:

- cooldown: Thời gian chờ sau khi sử dụng vũ khí trước khi có thể sử dụng lại (đơn vị: miligiây).
- damage: Sát thương cơ bản của từng loại vũ khí.
- graphic: Đường dẫn đến file ảnh vũ khí.

Để ta có thể tương tác được với nhân vật thì ở file player.py tiếp nối đoạn di chuyển của nhân vật ta thêm các dòng sau:

```
# attack input
if keys[pygame.K_SPACE]:
    self.attacking = True
    self.attack_time = pygame.time.get_ticks()
    self.create_attack()
    self.weapon_attack_sound.play()
```

HÌNH 2.22. TƯƠNG TÁC VỚI VŨ KHÍ

Đoạn code này xử lý hành vi tấn công của nhân vật trong game. Khi người chơi nhấn phím space:

- Nhân vật sẽ chuyển sang trạng thái tấn công.

- Thời gian bắt đầu tấn công được ghi nhận để bắt đầu quá trình cooldown vũ khí.
- Gọi đến phương thức create_attack và hình ảnh tấn công được tạo ra cùng với âm thanh.

```
def create_attack(self):
    self.current_attack = Weapon(self.player, [self.visible_sprites, self.attack_sprites])
```

HÌNH 2.23. PHƯƠNG THỨC CREATE_ATTACK

Phương thức này được gọi để tạo ra một đòn tấn công của nhân vật.

- Tham số đầu tiên (self.player) cung cấp thông tin về nhân vật
- Tham số thứ hai là một list chứa hai nhóm sprite, giúp hiển thị đòn tấn công trên màn hình và quản lý nó trong logic game.

2.4.3. Hệ thống phép thuật

```
# magic
magic_data = {
    'flame': {'strength': 10, 'cost': 10, 'graphic':'../graphics/particles/flame/fire.png'},
    'heal' : {'strength': 20, 'cost': 10, 'graphic':'../graphics/particles/heal/heal.png'}
}
```

HÌNH 2.24. DỮ LIỆU PHÉP THUẬT

magic_data là một dictionary dùng để lưu trữ thông tin về các loại vũ khí trong trò chơi, với mỗi key (khóa) là tên của một loại phép thuật (trong trò chơi là: 'flame' và 'heal') và value (giá trị) là một dictionary khác chứa các thuộc tính của loại phép thuật đó:

- strength: Biểu thị mức độ ảnh hưởng của phép thuật.
  - 'flame' có sát thương là 15.
  - 'heal' hồi phục 20 HP mỗi lần sử dụng.
- cost: Lượng mana cần thiết để sử dụng phép thuật.
  - 'flame' cần 20 mana.
  - 'heal' cần là 10 mana.
- graphic: Đường dẫn đến file ảnh đại diện của phép thuật.

Để nhân vật sử dụng được phép thuật, tiếp nối dòng tấn công đã viết ở trên ta lập trình như sau:

```
# magic input
if keys[pygame.K_LCTRL]:
    self.attacking = True
    self.attack_time = pygame.time.get_ticks()
    style = list(magic_data.keys())[self.magic_index]
    strength = list(magic_data.values())[self.magic_index]['strength'] + self.stats['magic']
    cost = list(magic_data.values())[self.magic_index]['cost']
    self.create_magic(style, strength, cost)
```

HÌNH 2.25. TƯƠNG TÁC VỚI PHÉP THUẬT

Đoạn code này xử lý hành vi sử dụng phép thuật của nhân vật trong game. Khi người chơi nhấn phím ctrl bên trái:

- Nhân vật sẽ chuyển sang trạng thái dùng phép thuật.
- Thời gian bắt đầu sử dụng phép thuật được ghi nhận để bắt đầu quá trình cooldown phép và tính toán lượng mana.
- Gọi đến phương thức create_magic và hình ảnh tấn công được tạo ra cùng với âm thanh.

```
def create_magic(self, style, strength, cost):
    if style == 'heal':
        self.magic_player.heal(self.player, strength, cost, [self.visible_sprites])

    if style == 'flame':
        self.magic_player.flame(self.player, cost, [self.visible_sprites, self.attack_sprites])
```

HÌNH 2.26. PHƯƠNG THÚC CREATE_MAGIC

Phương thức này được gọi để xử lý các thông tin khi nhân vật sử dụng phép thuật:

- Nếu phép dùng là "heal":
- self.player: tham chiếu đến đối tượng đại diện cho nhân vật.
- strength: lượng máu hồi phục đã thiết lập trong setiings.py.
- cost: lượng mana cần thiết để sử dụng phép thuật ‘heal’.
- [self.visible_sprites]: hiển thị hiệu ứng hồi máu.
- Nếu phép dùng là "flame":
- self.player: tham chiếu đến đối tượng đại diện cho nhân vật.
- strength: lượng sát thương phép đã thiết lập trong setiings.py.

- cost: lượng mana cần thiết để sử dụng phép thuật ‘flame’.
- Tham số tiếp theo là một list chứa hai nhóm sprite, giúp hiển thị lửa trên màn hình và quản lý nó trong logic game.

2.4.4. Logic game

```
def player_attack_logic(self):
    if self.attack_sprites:
        for attack_sprite in self.attack_sprites:
            collision_sprites = pygame.sprite.spritecollide(attack_sprite, self.attackable_sprites, False)
            if collision_sprites:
                for target_sprite in collision_sprites:
                    if target_sprite.sprite_type == 'grass':
                        pos = target_sprite.rect.center
                        offset = pygame.math.Vector2(0,75)
                        for leaf in range(randint(3,6)):
                            self.animation_player.create_grass_particles(pos - offset, [self.visible_sprites])
                        target_sprite.kill()
                    else:
                        target_sprite.get_damage(self.player, attack_sprite.sprite_type)
```

HÌNH 2.27. LOGIC TẤN CÔNG

Hàm player_attack_logic(self) mô phỏng logic tấn công của nhân vật chơi. Hàm này có chức năng xử lý logic tấn công của nhân vật.

- Kiểm tra đòn tấn công: Nếu attack_sprites không rỗng (nghĩa là nhân vật đang tấn công) thì sẽ tiến hành xử lý va chạm.
- Xử lý va chạm với từng đòn tấn công:
- Duyệt qua từng sprite trong danh sách attack_sprites.
- Sử dụng hàm pygame.sprite.spritecollide để kiểm tra va chạm giữa sprite tấn công với danh sách attackable_sprites (danh sách các sprite có thể bị tấn công).
- Tham số False của pygame.sprite.spritecollide có nghĩa là chỉ kiểm tra va chạm, không loại bỏ các sprite đã va chạm khỏi danh sách.
- Xử lý đối tượng bị va chạm:
- Nếu là ('grass'), tạo hiệu ứng lá bay lên.
- Xóa bỏ đối tượng (target_sprite.kill()).
- Nếu không phải ‘grass’ sẽ gây sát thương cho đối tượng bị va chạm.
- Sử dụng hàm get_damage (giả sử đây là hàm xử lý sát thương) của đối tượng bị va chạm (target_sprite) để tính toán và trừ đi lượng sát thương. Tham số truyền vào là nhân vật (self.player) và loại đòn tấn công (attack_sprite.sprite_type).

2.5. Xây dựng quái vật

HÌNH 2.28. CÁC LOẠI QUÁI TRONG TRÒ CHƠI

Đây là các quái vật của trò chơi. Để cho các quái vật có thể chuyển động cũng như tương tác với mọi thứ xung quanh, trong file settings.py ta thêm dữ liệu của các quái vật như hình dưới:

```
# enemy
monster_data = {
    'squid': {
        'health': 200,
        'exp': 100,
        'damage': 25,
        'attack_type':
        'slash',
        'attack_sound': '../audio/attack/slash.wav',
        'speed': 3,
        'resistance': 3,
        'attack_radius': 80,
        'notice_radius': 360
        },
    'raccoon': {'health': 1000, 'exp': 500, 'damage': 55, 'attack_type': 'claw', 'attack_sound': '../audio/attack/claw.wav', 'speed': 2,
    'spirit': {'health': 290, 'exp': 110, 'damage': 14, 'attack_type': 'thunder', 'attack_sound': '../audio/attack/fireball.wav', 'speed'
    'bamboo': {'health': 170, 'exp': 120, 'damage': 12, 'attack_type': 'leaf_attack', 'attack_sound': '../audio/attack/slash.wav', 'speed
}
```

HÌNH 2.29. DỮ LIỆU QUÁI VẬT

Đây là dictionary tên monster_data lưu trữ thông tin về các loại quái vật trong trò chơi. Mỗi key của dictionary là tên của một loại quái vật (ví dụ: 'squid', 'raccoon',…), value của mỗi key chứa các thuộc tính của quái vật đó.

- 'health': Máu của quái vật.

- 'exp': Số điểm nhận được khi tiêu diệt quái vật này.
- 'damage': Sát thương của quái vật.
- 'attack_type': Loại hình tấn công của quái vật ('slash' - chém, 'claw' - cào, 'thunder' - sé điện, 'leaf_attack' - tấn công bằng lá).
- 'attack_sound': Đường dẫn đến file âm thanh phát ra khi quái vật tấn công.
- 'speed': Tốc độ di chuyển của quái vật.
- 'resistance': Kháng của quái vật.
- 'attack_radius': Bán kính tấn công của quái vật.
- 'notice_radius': Bán kính phát hiện của quái.

Trong file enemy.py ta sẽ có các thiết lập sau:

```
class Enemy(Entity):
    def __init__(self, monster_name, pos, groups, obstacle_sprites, damage_player, trigger_death_particles, add_exp):

        # general setup
        super().__init__(groups)
        self.sprite_type = 'enemy'
```

HÌNH 2.30. KHỞI TẠO CLASS ENEMY

Đầu tiên là khởi tạo class enemy

```
# graphics setup
self.import_graphics(monster_name)
self.status = 'idle'
self.image = self.animations[self.status][self.frame_index]

# movement
self.rect = self.image.get_rect(topleft = pos)
self.hitbox = self.rect.inflate(0, -10)
self.obstacle_sprites = obstacle_sprites

# stats
self.monster_name = monster_name
monster_info = monster_data[self.monster_name]
self.health = monster_info['health']
self.exp = monster_info['exp']
self.speed = monster_info['speed']
self.attack_damage = monster_info['damage']
self.resistance = monster_info['resistance']
self.attack_radius = monster_info['attack_radius']
self.notice_radius = monster_info['notice_radius']
self.attack_type = monster_info['attack_type']
```

HÌNH 2.31. THUỘC TÍNH TRONG CLASS ENEMY (1)

Trong đó, class Enemy được xây dựng và định nghĩa các thuộc tính bao gồm tên, vị trí, sát thương, hình ảnh và điểm. Ta tiếp tục thiết lập các thuộc tính như hình ảnh hoạt ảnh, vùng va chạm, thông tin về chướng ngại vật, và các thuộc tính quan trọng của quái vật như máu, sát thương, tốc độ dựa vào dữ liệu được cung cấp ở trên.

```
# player interaction
self.can_attack = True
self.attack_time = None
self.attack_cooldown = 400
self.damage_player = damage_player
self.trigger_death_particles = trigger_death_particles
self.add_exp = add_exp

# invincibility timer
self.vulnerable = True
self.hit_time = None
self.invincibility_duration = 300
```

HÌNH 2.32. THUỘC TÍNH TRONG CLASS ENEMY (2)

Đoạn code này thiết lập các yếu tố như thời gian hồi chiêu tấn công, sát thương gây ra, hiệu ứng chết và điểm của quái vật. Bên cạnh đó, nó còn thiết lập cơ chế bất tử trong một khoảng thời gian ngắn để tránh trường hợp quái vật bị hạ gục quá nhanh.

```
# sounds
self.death_sound = pygame.mixer.Sound('../audio/death.wav')
self.hit_sound = pygame.mixer.Sound('../audio/hit.wav')
self.attack_sound = pygame.mixer.Sound(monster_info['attack_sound'])
self.death_sound.set_volume(0.4)
self.hit_sound.set_volume(0.4)
self.attack_sound.set_volume(0.4)
```

HÌNH 2.33. THUỘC TÍNH TRONG CLASS ENEMY (3)

Đoạn code này thiết lập các hiệu ứng âm thanh cho các hành động của quái.

Sau khi đã có các thiết lập của quái vật, tiếp theo ta sẽ làm cho các quái vật có thể tương tác được với nhân vật và địa hình trong trò chơi. Đầu tiên ta sẽ đưa hình ảnh của quái vào trò chơi.

```
def import_graphics(self, name):
    self.animations = {'idle': [], 'move': [], 'attack': []}
    main_path = f'../graphics/monsters/{name}/'
    for animation in self.animations.keys():
        self.animations[animation] = import_folder(main_path + animation)
```

HÌNH 2.34. PHƯƠNG THỨC IMPORT_GRAPHICS

Tiếp đến thêm phương thức tính toán khoảng cách và hướng di chuyển của quái đến nhân vật.

```
def get_player_distance_direction(self, player):
    enemy_vec = pygame.math.Vector2(self.rect.center)
    player_vec = pygame.math.Vector2(player.rect.center)
    distance = (player_vec - enemy_vec).magnitude()

    if distance > 0:
        direction = (player_vec - enemy_vec).normalize()
    else:
        direction = pygame.math.Vector2()

    return (distance, direction)
```

HÌNH 2.35. PHƯƠNG THỨC GET_PLAYER_DISTANCE_DIRECTION

Kế tiếp ta thêm trạng thái hoạt ảnh (nghỉ, di chuyển, tấn công) dựa trên khoảng cách đến nhân vật.

```
def get_status(self, player):
    distance = self.get_player_distance_direction(player)[0]

    if distance <= self.attack_radius and self.can_attack:
        if self.status != 'attack':
            self.frame_index = 0
        self.status = 'attack'
    elif distance <= self.notice_radius:
        self.status = 'move'
    else:
        self.status = 'idle'
```

HÌNH 2.36. PHƯƠNG THỨC GET_STATUS

Kèm theo trạng thái hoạt ảnh ta cần thêm hành động của quái vật.

```
def actions(self, player):
    if self.status == 'attack':
        self.attack_time = pygame.time.get_ticks()
        self.damage_player(self.attack_damage, self.attack_type)
        self.attack_sound.play()
    elif self.status == 'move':
        self.direction = self.get_player_distance_direction(player)[1]
    else:
        self.direction = pygame.math.Vector2()
```

HÌNH 2.37. PHƯƠNG THỨC ACTIONS

Để các phương thức vừa tạo trên hoạt động ta tạo thêm một phương thức animate. Phương thức này cập nhật hình ảnh hiển thị theo từng khung hình. Đồng thời, nó còn xử lý animatio (đặc biệt là animation tấn công) và tạo hiệu ứng bất tử.

```
def animate(self):
    animation = self.animations[self.status]

    self.frame_index += self.animation_speed
    if self.frame_index >= len(animation):
        if self.status == 'attack':
            self.can_attack = False
        self.frame_index = 0

    self.image = animation[int(self.frame_index)]
    self.rect = self.image.get_rect(center = self.hitbox.center)

    if not self.vulnerable:
        alpha = self.wave_value()
        self.image.set_alpha(alpha)
    else:
        self.image.set_alpha(255)
```

HÌNH 2.38. PHƯƠNG THỨC ANIMATE

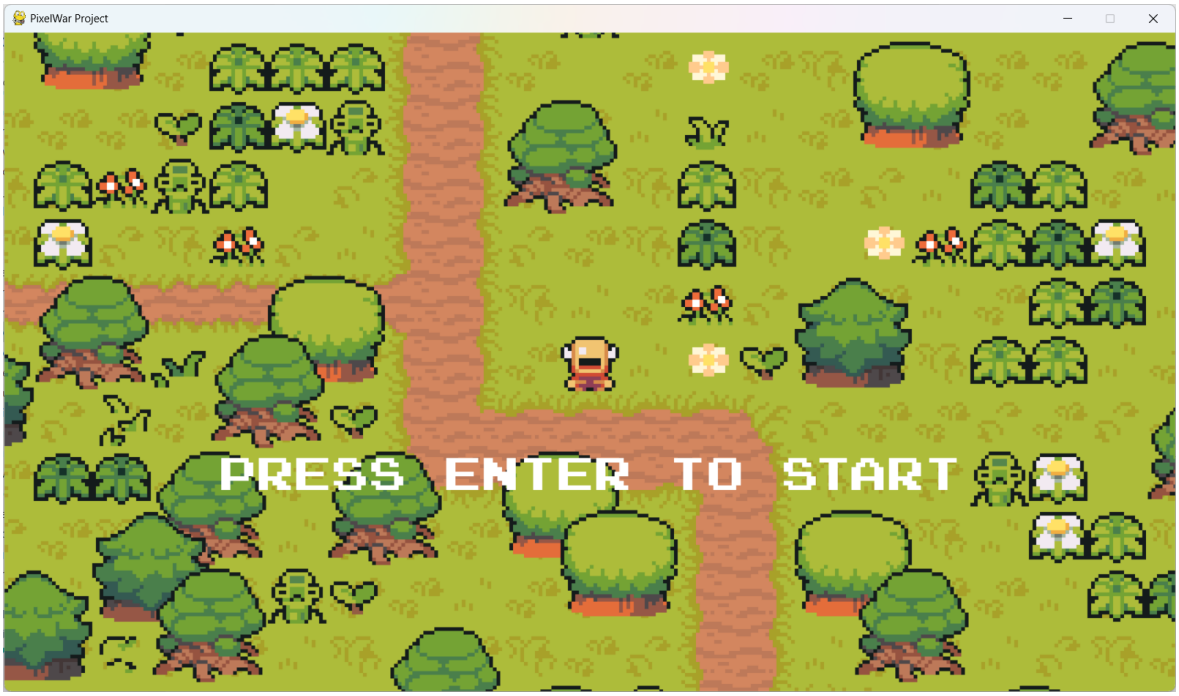
Cuối cùng ta thêm phương thức để thể hiện trạng thái bị đánh bại của nhân vật.

```
def check_death(self):
    if self.health <= 0:
        self.kill()
        self.trigger_death_particles(self.rect.center, self.monster_name)
        self.add_exp(self.exp)
        self.death_sound.play()
```

HÌNH 2.39. PHƯƠNG THỨC CHECK_DEATH

Chương 3. DEMO

3.1. Giao diện bắt đầu game



HÌNH 3.1. GIAO DIỆN BẮT ĐẦU GAME

Để bắt đầu trò chơi ta nhấn Enter như trên. Ta có các nút thao tác như sau:

- Di chuyển: Left, Right, Up, Down
- Tấn công: Space
- Tấn công bằng phép: Left Ctrl
- Nâng cấp thông số: M

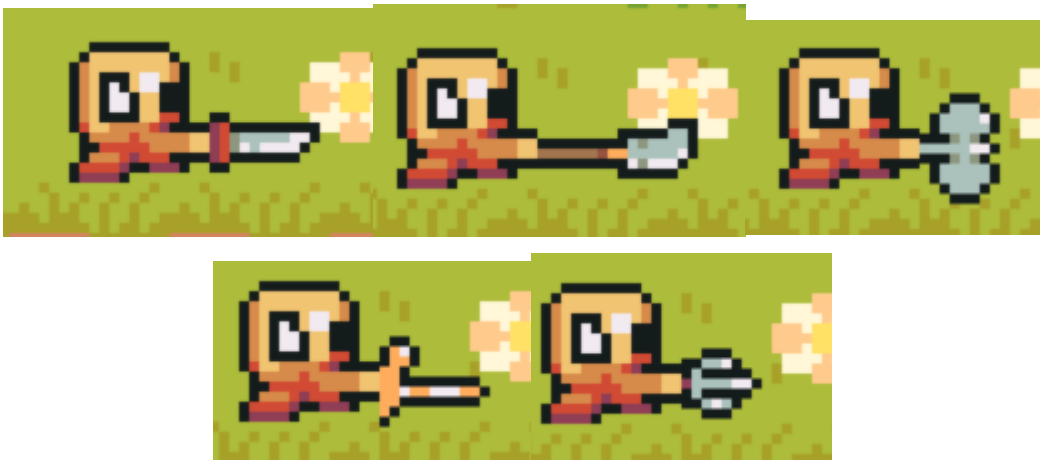
- Space để tăng
- Left or Right để di chuyển

3.2. Giao diện bị quái vật đánh bại

HÌNH 3.2. GIAO DIỆN BỊ ĐÁNH BẠI

3.3. Tấn công bằng vũ khí

Trong trò chơi này nhân vật có thể sử dụng được 5 loại vũ khí: Rìu, Thương, Kiếm mảnh cạnh sắc, kiếm Sai và kiếm bình thường. Trong đó kiếm bình thường sẽ là mặc định của nhân vật.



HÌNH 3.3. HÌNH ẢNH 5 LOẠI VŨ KHÍ CỦA NHÂN VẬT

Nếu ta muốn đổi vũ khí thì sẽ nhấn nút Q. Thứ tự bắt đầu từ kiếm bình thường > Thương > Rìu > Kiếm mảnh cạnh sắc > Kiếm Sai.

HÌNH 3.4. NHÂN VẬT TẤN CÔNG QUÁI VẬT

3.4. . Sử dụng phép thuật

Hai loại phép thuật của nhân vật là flame và heal, sẽ có biểu tượng như sau:

HÌNH 3.5. HÌNH ẢNH 2 LOẠI PHÉP THUẬT CỦA NHÂN VẬT

Nếu như muốn đổi phép thuật ta nhấn E. Thứ tự bắt đầu là flame > heal.

HÌNH 3.6. NHÂN VẬT DÙNG FLAME ĐỂ TẤN CÔNG LÊN QUÁI VẬT

3.5. Link source code

https://github.com/QuangVo11311/PixelWar-Project/tree/main/code

KẾT QUẢ ĐẠT ĐƯỢC VÀ HƯỚNG PHÁT TRIỂN

Kết quả đạt được

Đồ án đã tiến hành khảo sát các kiến thức cần biết về Python và khái niệm cơ bản thế nào là một game thể loại RPG. Trong đồ án này, nhóm đã so sánh Python và các ngôn ngữ lập trình khác để thấy được vì sao người mới học lập trình nên chọn Python. Song song đó nhóm cũng đã nghiên cứu thư viện Pygame để đưa ra các ưu điểm cũng như nhược điểm của thư viện này. Cuối cùng nhóm em đã tổng hợp lại các kiến thức và lập trình ra game Pixel War.

Trong quá trình thực hiện đồ án, nhóm đã cố gắng tham khảo các tài liệu liên quan bằng tiếng Việt lẫn tiếng Anh. Tuy nhiên do thời gian và trình độ có hạn nên không tránh khỏi những hạn chế và thiếu sót nhất định. Do vậy em thật sự mong muốn nhận được những góp ý cả về kiến thức chuyên môn lẫn cách trình bày.

Hướng phát triển

Việc lập trình game bằng Python sử dụng thư viện Pygame đã cho nhóm thấy một hướng phát triển mới cho từng người. Trong tương lai, nhóm muốn tự mình tối ưu và cập nhật thêm một số chức năng cho game Pixel War như: thêm các màn chơi tiếp theo, thêm nhân vật mới hay nâng cấp hệ thống quái trong game.

TÀI LIỆU THAM KHẢO

[1] Allen B. Downey, *"Think Python: How to think like a computer scientist"*, Green Tea Press, 2nd edition, 2015.

[2] Mark Lutz, *"Learning Python"*, O'Reilly Media, Inc., 5th edition, June 2013.

[3] Swaroop, "A Bite of Python", Time document create [2004] [Online]. Available: https://python.swaroopch.com/ [07/03/2024].

[5] Lê Quang Nhật và Bùi Tấn Lâm, "*Xây dựng trợ lý ảo bằng Python*", Trường Đại học Công nghệ thông tin và Truyền thông Việt – Hàn, Báo cáo đồ án, 1/2020.

[4] Võ Duy Tuấn, *"Python cơ bản…Rất là CƠ BẢN"*, ebook, 2021.