SiNGULAR

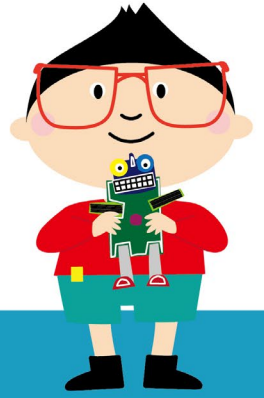
奇點創意

程 式 創 客 教 室

機器人 / AI人工智慧 / 程式語言

Join Singular!
Be a super inventor!

AnimeGames 12



Maker + Coder = Singular Super Inventor

物件化

- 某個RPG遊戲，裡面的職業有戰士、魔法師等
- 請大家觀察一下這些腳色相同與不同之處

物件化

- 遊戲中會有很多玩家，要如何針對每一位玩家都有體力、攻擊力、防禦力等屬性
- 如何快速針對不同玩家進行各別的基礎屬性設定呢?

物件化

```
# 基本玩家類別
class Player:
    def __init__(self, name, health, attack, defense):
        """初始化玩家, name: 名稱, health: 血量, attack: 攻擊, defense: 防禦"""
        # self的意思是創造一個屬於自己的變數
        self.name = name
        self.health = health
        self.attack = attack
        self.defense = defense
```

物件化

- 新增角色:

```
# 基本玩家類別
class Player:
    ...省略...

# 新增一個玩家
player1 = Player("你在哈囉", 100, 2, 9)
print(f"玩家名稱: {player1.name}")
print(f"玩家血量: {player1.health}")
print(f"玩家攻擊: {player1.attack}")
print(f"玩家防禦: {player1.defense}")

# 新增一個玩家
player2 = Player("你好啊", 50, 10, 5)
print(f"玩家名稱: {player2.name}")
print(f"玩家血量: {player2.health}")
print(f"玩家攻擊: {player2.attack}")
print(f"玩家防禦: {player2.defense}")
```

新增角色指令

- 如果每一位玩家都有被攻擊的情況發生那我們可以直接新增被攻擊的事件指令

```
# 基本玩家類別
class Player:
    def __init__(self, name, health, attack, defense):
        """初始化玩家, name: 名稱, health: 血量, attack: 攻擊, defense: 防禦"""
        # self的意思是創造一個屬於自己的變數
        self.name = name
        self.health = health
        self.attack = attack
        self.defense = defense

    def take_damage(self, damage):
        """受到傷害"""
        if damage > self.defense:
            self.health -= damage - self.defense
            return f"{self.name} 受到了 {damage} 點傷害！"
        else:
            return f"{self.name} 成功抵擋攻擊!!"
```

當玩家遭遇攻擊時

```
# 新增一個玩家
player1 = Player("你在哈囉", 100, 2, 9)
print(f"玩家名稱: {player1.name}")
print(f"玩家血量: {player1.health}")
print(f"玩家攻擊: {player1.attack}")
print(f"玩家防禦: {player1.defense}")

# 新增一個玩家
player2 = Player("你好啊", 50, 10, 5)
print(f"玩家名稱: {player2.name}")
print(f"玩家血量: {player2.health}")
print(f"玩家攻擊: {player2.attack}")
print(f"玩家防禦: {player2.defense}")

# 玩家1攻擊玩家2
print(player2.take_damage(player1.attack))
print(f"玩家2血量剩餘: {player2.health}")
```

物件化：繼承

- 當每一位玩家有不同的職業與特殊技能時我們可以透過繼承來創造出具有基本屬性且帶有不同特殊技能的職業物件

物件化：繼承

```
# 基本玩家類別
class Player:
    …省略…

# 法師類別
class Mage(Player):  # 繼承Player類別
    def __init__(self, name, health, attack, defense, magic_power):
        """初始化法師, name: 名稱, health: 血量, attack: 攻擊, defense: 防禦, magic_power: 魔力"""
        super().__init__(name, health, attack, defense)
        self.magic_power = magic_power

    def cast_spell(self):
        """施放魔法"""
        self.magic_power -= 10 # 消耗魔力，施放魔法攻擊
        return self.attack + self.magic_power
```

物件化:繼承

- 試看看，創造一個戰士的物件(Warrior)，並繼承基本玩家物件(Player)，新增一個裝甲屬性(armor)以及使用裝甲回復HP技能 use_armor

物件化：繼承

```
# 戰士類別
class Warrior(Player):  # 繼承Player類別
    def __init__(self, name, health, attack, defense, armor):
        """初始化戰士, name: 名稱, health: 血量, attack: 攻擊, defense: 防禦, armor: 裝甲"""
        super().__init__(name, health, attack, defense)
        self.armor = armor

    def use_armor(self):
        self.health += self.armor
        return f"{self.name} 使用裝甲，增加了 {self.armor} 點體力！"
```

物件化

- 繼承後會保有原本的物件屬性及方法

```
class Warrior(Player):  # 繼承Player類別
        …省略…

player1 = Warrior("戰士小明", 100, 15, 10, 5)
player2 = Mage("法師小華", 80, 10, 5, 20)

print(f"{player1.name}血量剩餘: {player1.health}")
print(player1.use_armor())
print(f"{player1.name}血量剩餘: {player1.health}")

print(f"{player2.name}目前魔力: {player2.magic_power}")
player1.take_damage(player2.cast_spell())
print(f"{player2.name}對{player1.name}施放魔法攻擊！")
print(f"{player2.name}目前魔力: {player2.magic_power}")
print(f"{player1.name}血量剩餘: {player1.health}")
```

回到小恐龍

- 當今天我們有2種障礙物(仙人掌、翼龍)時，請觀察哪一些是他們的基本屬性呢?
- 新增障礙物物件:

```
#########################障礙物物件##########################
class Obstacle:
    def __init__(self, x, y, img: list[pygame.Surface], shift):
        self.x = x
        self.y = y
        self.img = img
        self.shift = shift
        self.center_x = x + img[0].get_width() / 2
        self.center_y = y + img[0].get_height() / 2
        self.detect_r = max(img[0].get_width(), img[0].get_height()) / 2
        self.index = 0
```

障礙物物件

```
########################障礙物物件#########################
class Obstacle:
    def __init__(self, x, y, img, shift):
        …省略…
    def initial(self):
        self.x = bg_x - 100
        self.center_x = self.x + self.img[0].get_width() / 2
        self.center_y = self.y + self.img[0].get_height() / 2
        self.index = 0

    def move(self):
        self.x = (self.x - self.shift) % (bg_x - 100)
        self.index = (self.index - 1) % len(self.img)
        self.center_x = self.x + self.img[self.index].get_width() / 2
        self.center_y = self.y + self.img[self.index].get_height() / 2
        screen.blit(self.img[self.index], (self.x, self.y))
```

仙人掌物件:繼承

```
########################障礙物物件##########################
class Obstacle:
    …省略…
class Cacti(Obstacle):
    def __init__(self, x: int, y: int, img: list[pygame.Surface], shift: int):
        """初始化障礙物, x: x位置, y: y位置, img: 圖片, shift: 移動量"""
        super().__init__(x, y, img, shift)
        self.detect_r = self.detect_r - 15
```

翼龍物件：繼承

- 試試看寫一個翼龍物件並繼承障礙物物件

翼龍物件：繼承

- 試試看寫一個翼龍物件並繼承障礙物物件

```
##########################障礙物物件##########################
class Obstacle:
    …省略…
class Cacti(Obstacle):
    …省略…
class Ptera(Obstacle):
    def __init__(self, x: int, y: int, img: list, shift: int):
        """初始化障礙物, x: x位置, y: y位置, img: 圖片, shift: 移動量"""
        super().__init__(x, y, img, shift)
        self.detect_r = self.detect_r - 10
```

刪除指令

```
# def move_cacti():
# def move_ptera():
```

召喚物件

```
########################仙人掌物件##########################
# cacti_x = bg_x - 100  # 障礙物x位置
# cacti_y = LIMIT_LOW  # 障礙物y位置
# cacti_shift = 10  # 仙人掌移動量
# cacti_center_x = cacti_x + img_cacti.get_width() / 2  # 障礙物中心x位置
# cacti_center_y = cacti_y + img_cacti.get_height() / 2  # 障礙物中心y位置
# cacti_detect_r = max(img_cacti.get_width(), img_cacti.get_height()) / 2 - 15  # 障礙物偵測半徑

cacti = Cacti(bg_x - 100, LIMIT_LOW, [img_cacti], 10)
```

召喚物件

```
########################翼龍物件############################
# ptera_x = bg_x - 100  # 障礙物x位置
# ptera_y = PTERA_LIMIT_LOW  # 障礙物y位置
# ptera_index = 0  # 翼龍圖片編號
# ptera_shift = 10  # 翼龍移動量
# ptera_center_x = ptera_x + img_ptera[0].get_width() / 2  # 翼龍中心x位置
# ptera_center_y = ptera_y + img_ptera[0].get_height() / 2  # 翼龍中心y位置
# ptera_detect_r = max(img_ptera[0].get_width(), img_ptera[0].get_height()) / 2 - 10  # 翼龍偵測半徑

ptera = Ptera(bg_x - 100, PTERA_LIMIT_LOW, img_ptera, 10)
```

更新事件偵測

```
########################循環偵測#########################
while True:
    clock.tick(20)

    for event in pygame.event.get():
        …省略…
        if event.type == KEYDOWN:
                …省略…
            if event.key == K_RETURN and gg:
                score = 0
                gg = False
                # cacti_x = bg_x - 100
                # ptera_x = bg_x - 100
                ds_y = LIMIT_LOW
                jumpState = False
                cacti.initial()
                ptera.initial()
```

更新主程式

```
############################循環偵測############################
while True:
    …省略…
    if gg:
        game_over()
    else:
        …省略…
        score_update()
        if cacti.x <= 0 or ptera.x <= 0:
            score += 1
```

更新主程式

```
######################循環偵測#######################
while True:
        …省略…
        if cacti.x <= 0 or ptera.x <= 0:
            score += 1

        # cacti.move()
        # gg = is_hit(ds_center_x, ds_center_y, cacti.center_x, cacti.center_y, cacti.detect_r + ds_detect_r)

        ptera.move()
        gg = is_hit(ds_center_x, ds_center_y, ptera.center_x, ptera.center_y, ptera.detect_r + ds_detect_r)

        # if enemy_random == 0:
        #     move_cacti()
        #     gg = is_hit(ds_center_x, ds_center_y, cacti_center_x, cacti_center_y, cacti_detect_r + ds_detect_r)
        # else:
        #     move_ptera()
        #     gg = is_hit(ds_center_x, ds_center_y, ptera_center_x, ptera_center_y, ptera_detect_r + ds_detect_r)
    pygame.display.update()
```