



EXAMEN FINAL OPTATIVA I

PYTHON

DOCUMENTACION DEL CÓDIGO

NOMBRE: VICTOR GIMÉNEZ OJEDA

DOCENTE: ING. RICARDO MAIDANA

SEMESTRE: 9NO

CAACUPÉ – PARAGUAY

2024

1. Introducción

1.1 Propósito del Programa

El programa es un juego llamado "Guerra Espacial" desarrollado en Python utilizando la biblioteca Pygame. El objetivo del juego es romper bloques con una pelota controlada por una barra.

1.2 Requisitos

Python 3.x

Pygame

2. Estructura del Código

2.1 Importación de Bibliotecas

```
import pygame
```

```
import sys
```

```
import random
```

```
import pickle
```

Estas importaciones permiten utilizar funcionalidades de Pygame, manejo del sistema, generación de números aleatorios y almacenamiento de datos en archivos.

2.2 Inicialización y Configuración

```
pygame.init()
```

```
ANCHO, ALTO = 800, 600
```

```
NEGRO = (0, 0, 0)
```

```
BLANCO = (255, 255, 255)
```

```
ROJO = (255, 0, 0)
```

```
AZUL = (0, 0, 255)
```

```
VERDE = (0, 255, 0)
```

```
GRADIENTE_INICIAL = [(0, 0, 255), (0, 255, 255)]
```

Estas líneas inicializan Pygame y definen constantes para el tamaño de la pantalla y los colores utilizados en el juego.

2.3 Configuración de la Pantalla y Recursos

```
pantalla = pygame.display.set_mode((ANCHO, ALTO))
```

```
pygame.display.set_caption("Ani Ho'a Pelota")
```

Se configura la pantalla del juego y se establece el título de la ventana.

2.4 Fuentes y Sonidos

```
fuente_grande = pygame.font.SysFont("Arial", 60)
```

```
fuente_pequena = pygame.font.SysFont("Arial", 36)
```

```
sonido_pelota = pygame.mixer.Sound("pelota.wav")
```

```
sonido_perder = pygame.mixer.Sound("perder.wav")
```

```
sonido_romper = pygame.mixer.Sound("romper.mp3")
```

```
pygame.mixer.music.load("musica_fondo.wav")
```

```
pygame.mixer.music.play(-1)
```

```
pygame.mixer.music.set_volume(0.1)
```

Se cargarán las fuentes y los sonidos necesarios para el juego, incluyendo la música de fondo y los efectos de sonido.

2.5 Variables de Estado del Juego

```
INICIO = 0
```

```
JUGANDO = 1
```

```
PAUSA = 2
```

```
PERDIDO = 3
```

```
GANADO = 4
```

```
estado = INICIO
```

```
puntuacion = 0
```

```
vidas = 3
```

```
nivel = 1
```

```
velocidad_base = 5
```

```
incremento_velocidad = 1
```

Se definen los diferentes estados del juego y las variables que almacenan la puntuación, las vidas, el nivel y la velocidad de la pelota.

2.6 Cargar Imágenes

```
imagen_barra = pygame.image.load("barra.png").convert_alpha()
```

```
imagen_pelota = pygame.image.load("pelota_futbol.png").convert_alpha()
```

Se cargan las imágenes para la barra y la pelota.

3. Funciones

3.1 Funciones para Guardar y Cargar Puntuaciones

```
def guardar_puntuaciones(puntuaciones):  
    with open("puntuaciones.pkl", "wb") as archivo:  
        pickle.dump(puntuaciones, archivo)  
  
def cargar_puntuaciones():  
    try:  
        with open("puntuaciones.pkl", "rb") as archivo:  
            return pickle.load(archivo)  
    except FileNotFoundError:  
        return []
```

Estas funciones permiten guardar y cargar las evaluaciones del juego utilizando archivos pickle.

4. Clases

4.1 Clase Barra

```
class Barra(pygame.sprite.Sprite):  
    def __init__(self):  
        super().__init__()  
        self.image = pygame.transform.scale(imagen_barra, (150, 30))  
        self.rect = self.image.get_rect()  
        self.rect.x = ANCHO // 2 - self.rect.width // 2  
        self.rect.y = ALTO - 40  
        self.velocidad_x = 0  
  
    def update(self):
```

```
self.rect.x += self.velocidad_x
```

```
if self.rect.left < 0:
```

```
    self.rect.left = 0
```

```
if self.rect.right > ANCHO:
```

```
    self.rect.right = ANCHO
```

La clase Barra representa la barra controlada por el jugador. Se puede mover horizontalmente y tiene límites en los bordes de la pantalla.

4.2 Clase Bola

```
class Bola(pygame.sprite.Sprite):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.image = pygame.transform.scale(imagen_pelota, (25, 25))
```

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

```
        self.mask = pygame.mask.from_surface(self.image)
```

```
        self.rect.x = ANCHO // 2 - self.rect.width // 2
```

```
        self.rect.y = ALTO // 2 - self.rect.height // 2
```

```
        self.velocidad_x = velocidad_base
```

```
        self.velocidad_y = -velocidad_base
```

```
    def update(self):
```

```
        self.rect.x += self.velocidad_x
```

```
        self.rect.y += self.velocidad_y
```

```
        if self.rect.left <= 0 or self.rect.right >= ANCHO:
```

```
            self.velocidad_x *= -1
```

```
        if self.rect.top <= 0:
```

```
            self.velocidad_y *= -1
```

La clase Bola representa la pelota del juego, que rebota en los bordes de la pantalla y cambia de dirección al chocar con la barra y los bloques.

4.3 Clase Bloque

```
class Bloque(pygame.sprite.Sprite):  
  
    def __init__(self, x, y):  
  
        super().__init__()   
  
        self.color = random.choice(COLORES_BLOQUES)  
  
        self.image = pygame.Surface((75, 30))  
  
        self.image.fill(self.color)  
  
        self.rect = self.image.get_rect()  
  
        self.rect.x = x  
  
        self.rect.y = y
```

La clase Bloque representa los bloques que la pelota debe romper. Cada bloque tiene un color y una posición definida.

5. Funciones Auxiliares

5.1 Crear Bloques

```
def crear_bloques(nivel):  
  
    bloques = pygame.sprite.Group()  
  
    for fila in range(5 + nivel):  
  
        for columna in range(10):  
  
            bloque = Bloque(10 + columna * 78, 10 + fila * 35)  
  
            bloques.add(bloque)  
  
    return bloques
```

Esta función crea un grupo de bloques basado en el nivel actual del juego.

5.2 Dibujar Degradado

```
def dibujar_gradiente(superficie, color1, color2):
    for y in range(ALTO):
        color = (
            color1[0] + (color2[0] - color1[0]) * y // ALTO,
            color1[1] + (color2[1] - color1[1]) * y // ALTO,
            color1[2] + (color2[2] - color1[2]) * y // ALTO
        )
        pygame.draw.line(superficie, color, (0, y), (ANCHO, y))
```

Dibuja un gradiente vertical en la pantalla.

5.3 Dibujar Botón

```
def dibujar_boton(texto, fuente, color, x, y, ancho, alto):
    superficie_boton = pygame.Surface((ancho, alto))
    superficie_boton.fill(color)
    texto_boton = fuente.render(texto, True, BLANCO)
    pantalla.blit(superficie_boton, (x, y))
    pantalla.blit(texto_boton, (x + (ancho - texto_boton.get_width()) // 2, y + (alto -
    texto_boton.get_height()) // 2))
    return pygame.Rect(x, y, ancho, alto)
```

Dibuja un botón en la pantalla con texto centrado.

6. Funciones de Dibujo de Pantallas

6.1 Pantalla de Inicio

```
def dibujar_pantalla_inicio():
    dibujar_gradiente(pantalla, *GRADIENTE_INICIAL)
    texto_inicio = fuente_grande.render("ANI HO'A PELOTA", True, BLANCO)
```



```

    pantalla.blit(texto_inicio, (ANCHO // 2 - texto_inicio.get_width() // 2, ALTO // 4
- texto_inicio.get_height() // 2))

    boton_inicio = dibujar_boton("Iniciar Juego", fuente_pequena, ROJO, ANCHO
// 2 - 100, ALTO // 2, 200, 50)

    boton_salir = dibujar_boton("Salir", fuente_pequena, ROJO, ANCHO // 2 - 100,
ALTO // 2 + 60, 200, 50)

    texto_sub = fuente_pequena.render("o puedes presionar ENTER para iniciar
el juego", True, BLANCO)

    pantalla.blit(texto_sub, (ANCHO // 2 - texto_sub.get_width() // 2, ALTO // 2 +
130))

    texto_puntuaciones = fuente_pequena.render("Top 5 Puntuaciones Altas:",
True, BLANCO)

    pantalla.blit(texto_puntuaciones, (10, 10))

    for i, punt in enumerate(puntuaciones):

        texto = fuente_pequena.render(f"{i+1}. {punt}", True, BLANCO)

        pantalla.blit(texto, (10, 40 + i * 30))

    return boton_inicio, boton_salir

```

Dibuja la pantalla de inicio del juego con el título, botones y evaluaciones más altas.

6.2 Pantalla de Pausa

```

def dibujar_pantalla_pausa():

    texto_pausa = fuente_grande.render("PAUSA", True, BLANCO)

    pantalla.blit(texto_pausa, (ANCHO // 2 - texto_pausa.get_width() // 2, ALTO //
2 - texto_pausa.get_height() // 2))

    texto_sub = fuente_pequena.render("Presiona ESPACIO para continuar",
True, BLANCO)

```

```
pantalla.blit(texto_sub, (ANCHO // 2 - texto_sub.get_width() // 2, ALTO // 2 + 60))
```

Dibuja la pantalla de pausa con un mensaje de pausa y una instrucción para continuar.

6.3 Pantalla de Perdido

```
def dibujar_pantalla_perdido():
```

```
    texto_perdido = fuente_grande.render("PERDISTE", True, BLANCO)
```

```
    pantalla.blit(texto_perdido, (ANCHO // 2 - texto_perdido.get_width() // 2, ALTO // 2 - texto_perdido.get_height() // 2))
```

```
    texto_sub = fuente_pequena.render("Presiona ENTER para reiniciar", True, BLANCO)
```

```
    pantalla.blit(texto_sub, (ANCHO // 2 - texto_sub.get_width() // 2, ALTO // 2 + 60))
```

Dibuja la pantalla de pérdida con un mensaje de derrota y una instrucción para reiniciar.

6.4 Pantalla de Ganado

```
def dibujar_pantalla_ganado():
```

```
    texto_ganado = fuente_grande.render("¡GANASTE!", True, BLANCO)
```

```
    pantalla.blit(texto_ganado, (ANCHO // 2 - texto_ganado.get_width() // 2, ALTO // 2 - texto_ganado.get_height() // 2))
```

```
    texto_sub = fuente_pequena.render("Presiona ENTER para reiniciar", True, BLANCO)
```

```
    pantalla.blit(texto_sub, (ANCHO // 2 - texto_sub.get_width() // 2, ALTO // 2 + 60))
```

Dibuja la pantalla de victoria con un mensaje de triunfo y una instrucción para reiniciar.

7. Bucle Principal del Juego

El bucle principal del juego se encarga de gestionar el estado del juego, actualizar los sprites y gestionar los eventos del usuario.

while True:

```
    for evento in pygame.event.get():
```

```
        if evento.type == pygame.QUIT:
```

```
            pygame.quit()
```

```
            sys.exit()
```

```
        elif evento.type == pygame.KEYDOWN:
```

```
            if estado == INICIO and evento.key == pygame.K_RETURN:
```

```
                estado = JUGANDO
```

```
            elif estado == PAUSA and evento.key == pygame.K_SPACE:
```

```
                estado = JUGANDO
```

```
            elif estado == PERDIDO and evento.key == pygame.K_RETURN:
```

```
                estado = INICIO
```

```
            elif estado == GANADO and evento.key == pygame.K_RETURN:
```

```
                estado = INICIO
```

```
    if estado == INICIO:
```

```
        boton_inicio, boton_salir = dibujar_pantalla_inicio()
```

```
        for evento in pygame.event.get():
```

```
            if evento.type == pygame.MOUSEBUTTONDOWN:
```

```
                if boton_inicio.collidepoint(evento.pos):
```

```
                    estado = JUGANDO
```

```
                elif boton_salir.collidepoint(evento.pos):
```

```
                    pygame.quit()
```

```
        sys.exit()

elif estado == JUGANDO:

    pantalla.fill(NEGRO)

    # Actualizar y dibujar todos los sprites del juego

elif estado == PAUSA:

    dibujar_pantalla_pausa()

elif estado == PERDIDO:

    dibujar_pantalla_perdido()

elif estado == GANADO:

    dibujar_pantalla_ganado()

pygame.display.flip()

pygame.time.Clock().tick(60)
```

8. Conclusión

El código del juego "Guerra Espacial" está estructurado para manejar diferentes estados de juego y actualizar los elementos de la pantalla en consecuencia. La interacción del usuario se gestiona a través de eventos de teclado y ratón, proporcionando una experiencia de juego fluida y reactiva.

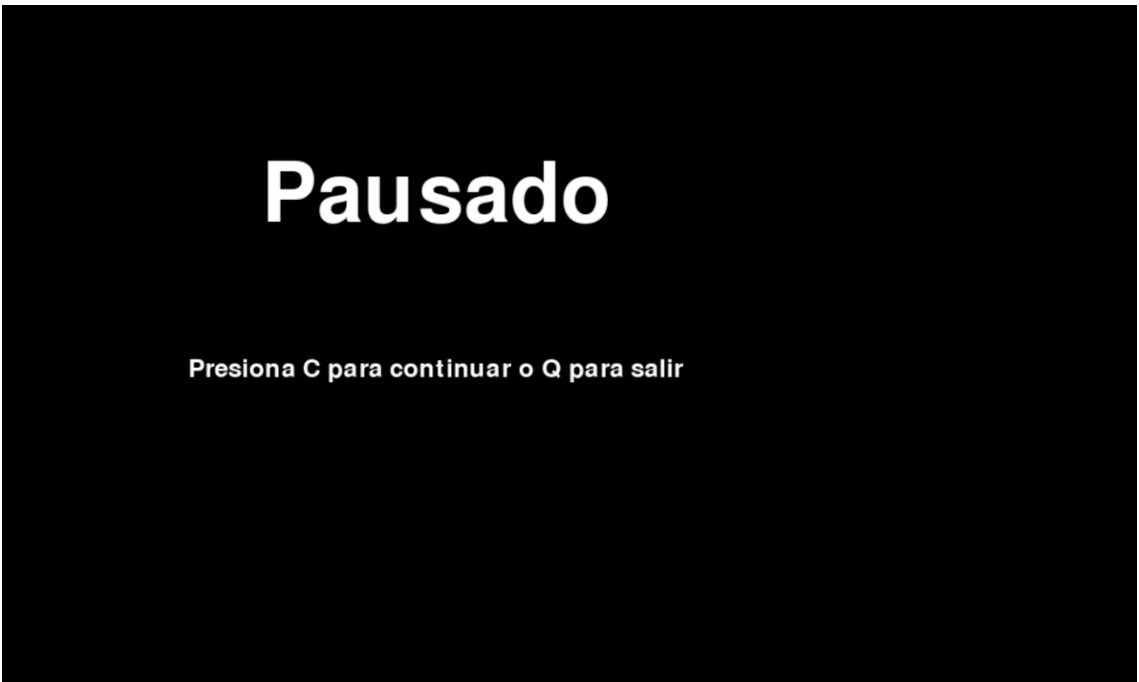
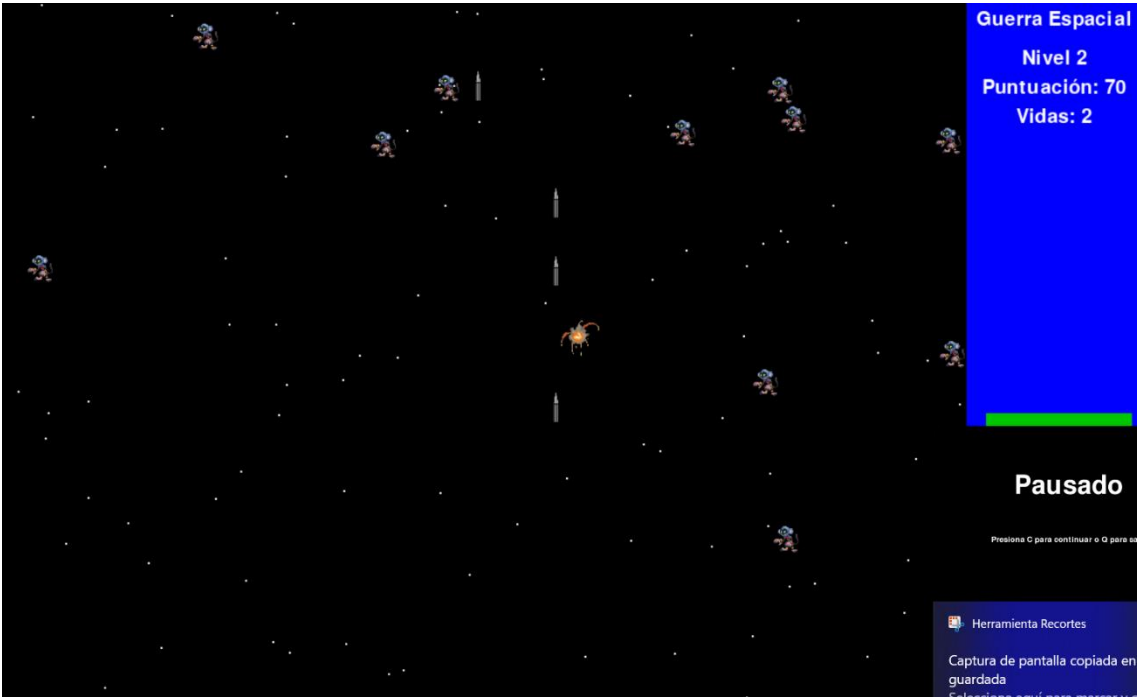
9. Anexo

Felicidades...Has superado todos los niveles!!

Salir

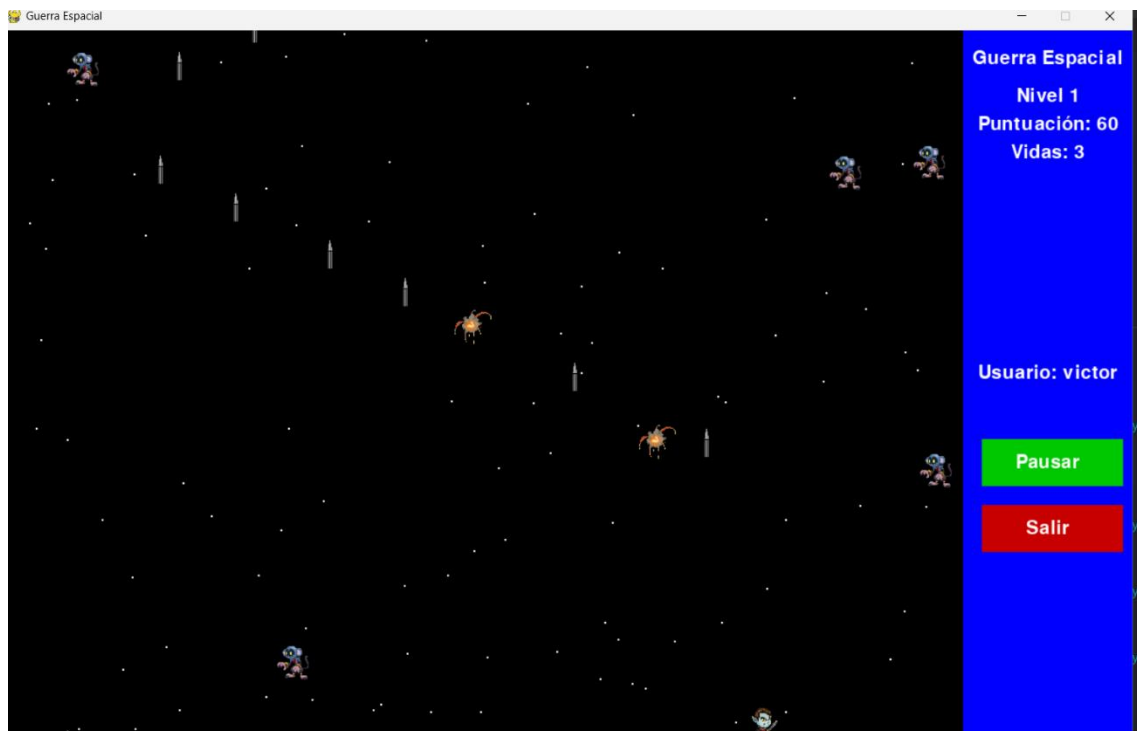
Felicidades!!! Segundo Nivel Superado!!!

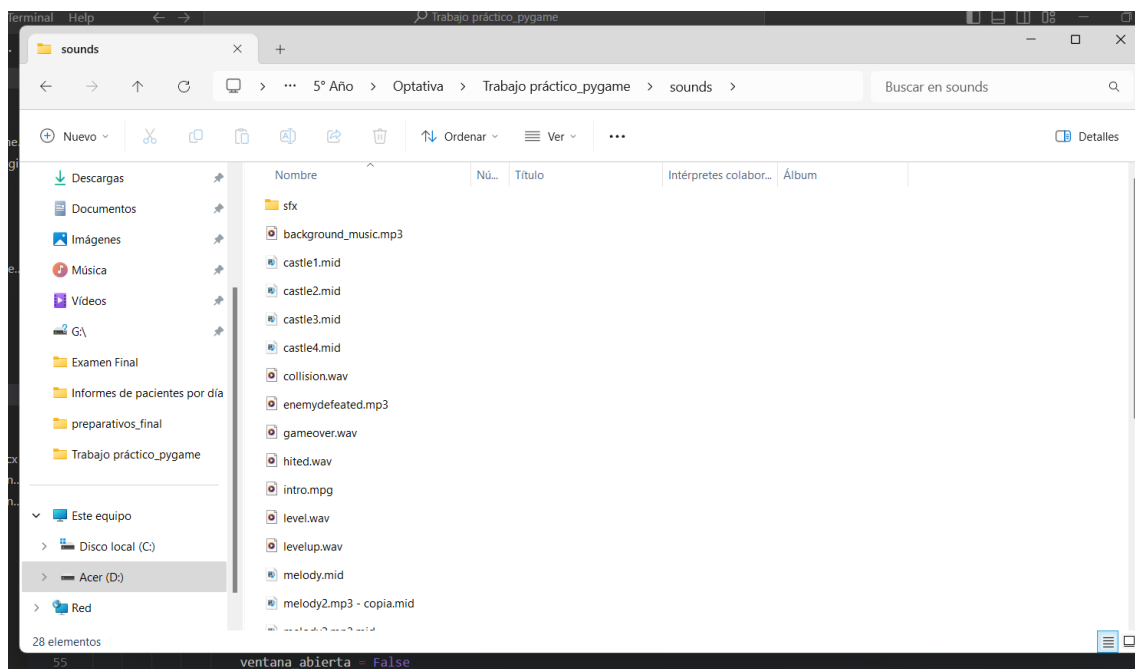
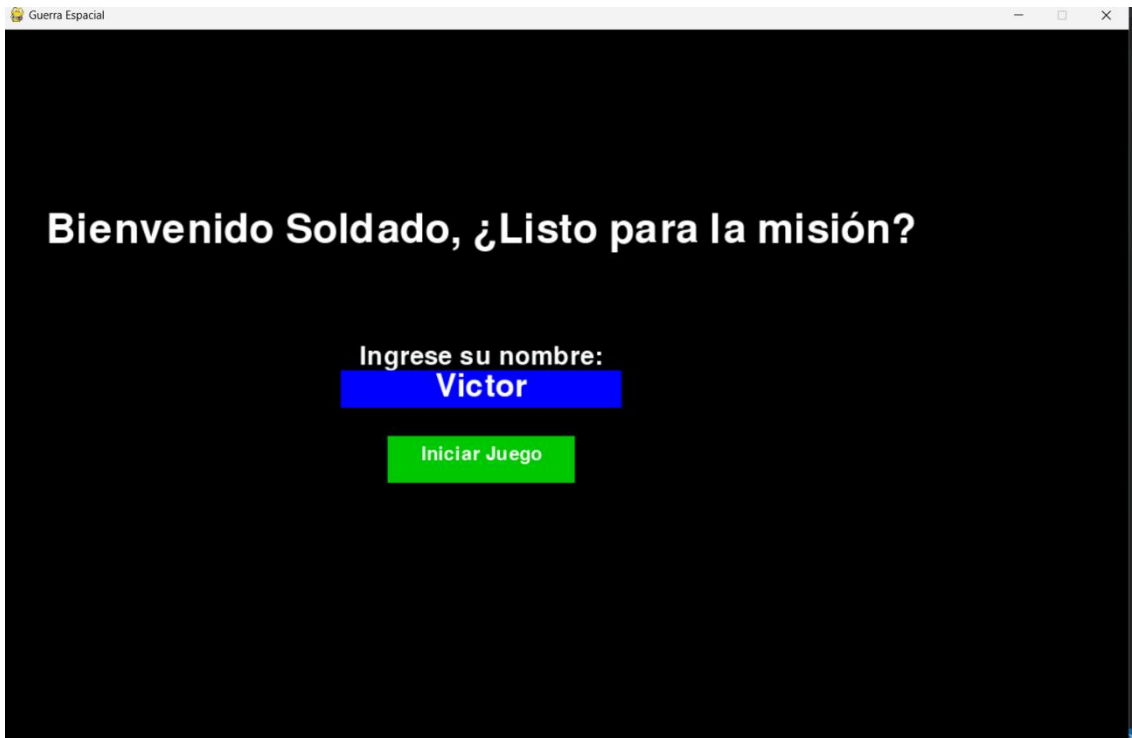
Tercer nivel

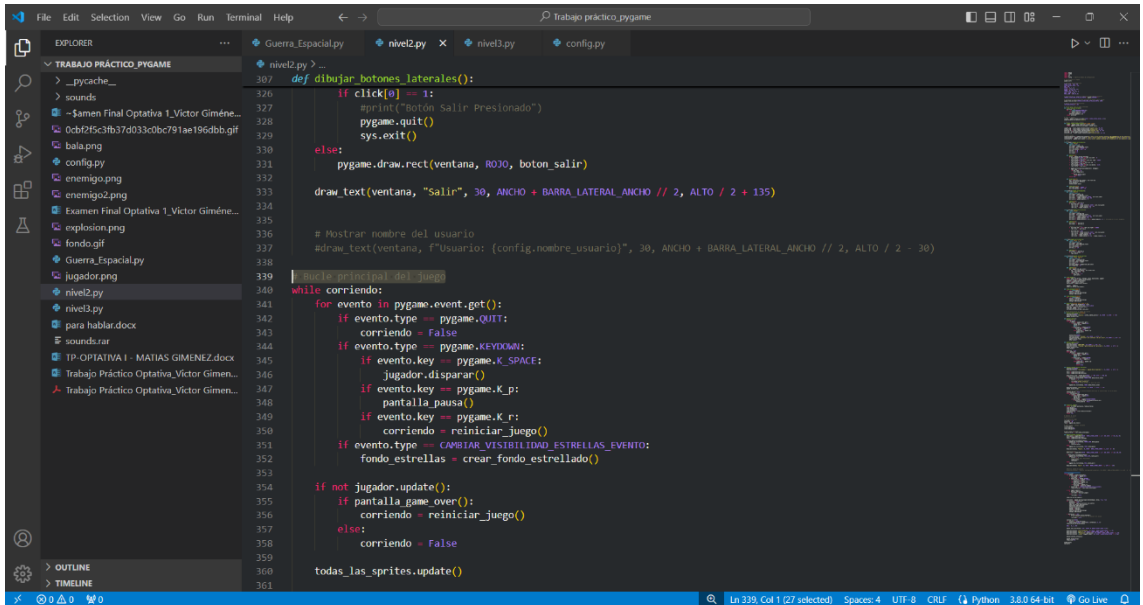
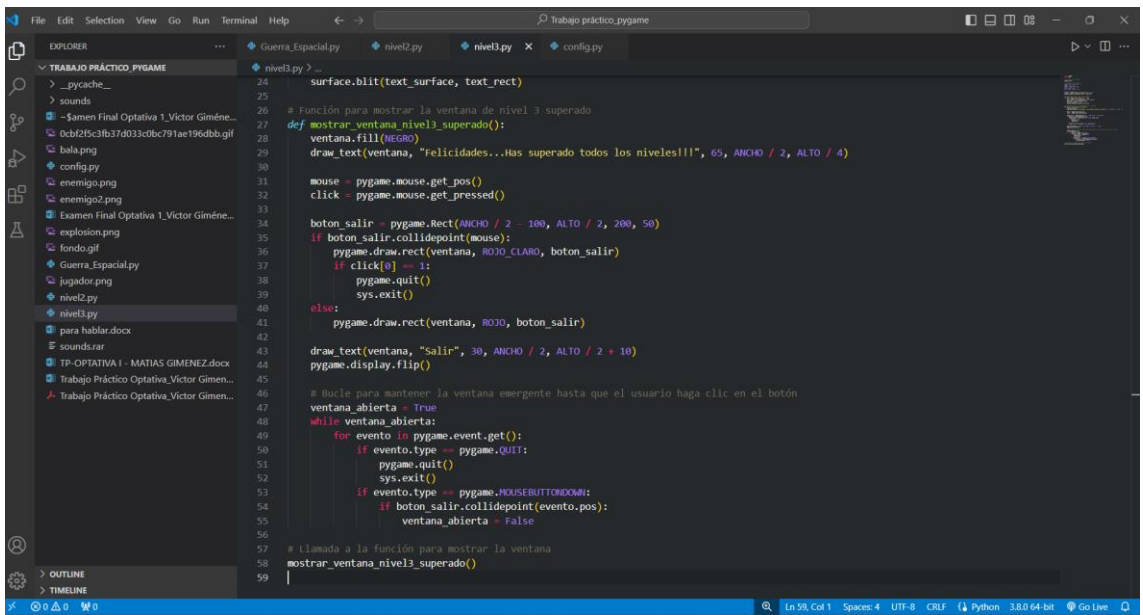
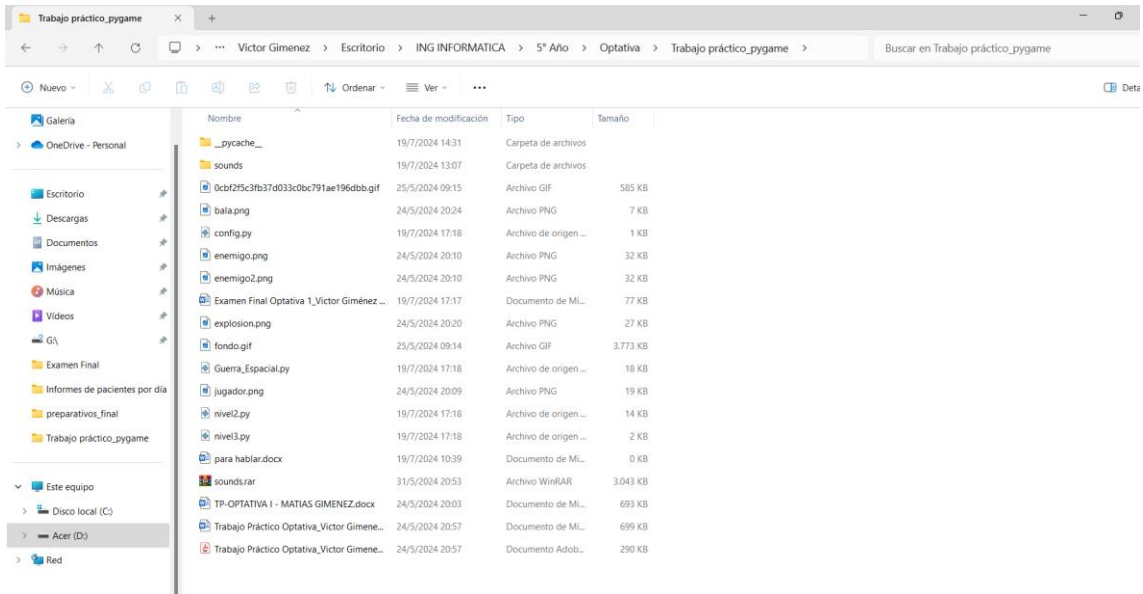


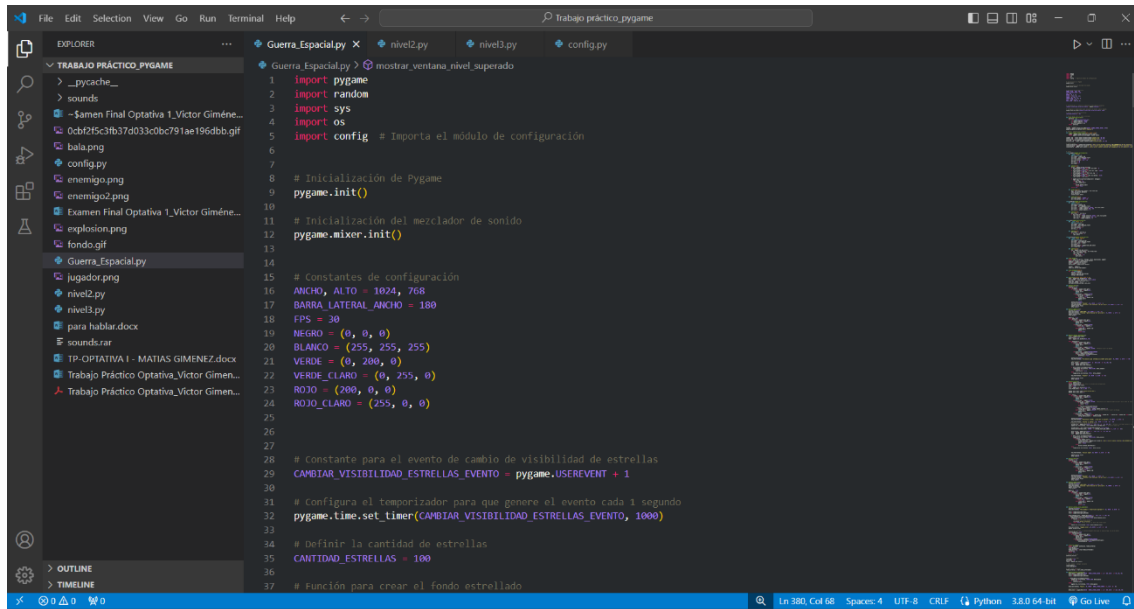
Felicidades!!! Primer Nivel Superado!!!

Segundo nivel

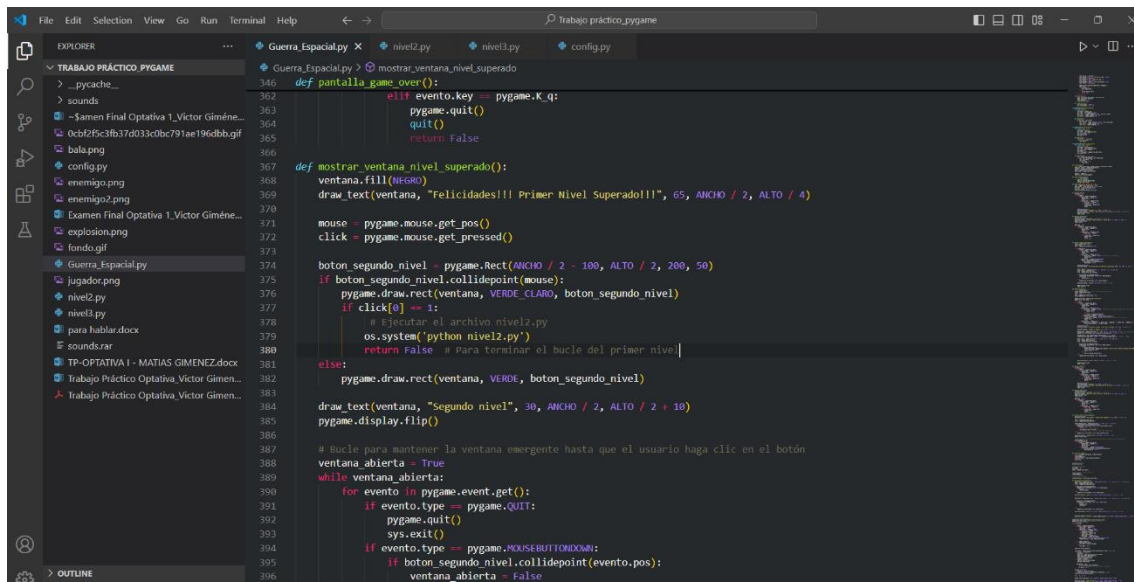








```
1 import pygame
2 import random
3 import sys
4 import os
5 import config # Importa el módulo de configuración
6
7
8 # Inicialización de Pygame
9 pygame.init()
10
11 # Inicialización del mezclador de sonido
12 pygame.mixer.init()
13
14
15 # Constantes de configuración
16 ANCHO, ALTO = 1024, 768
17 BARRA_LATERAL_ANCHO = 180
18 FPS = 30
19 NEGRO = (0, 0, 0)
20 BLANCO = (255, 255, 255)
21 VERDE = (0, 280, 0)
22 VERDE_CLARO = (0, 255, 0)
23 ROJO = (280, 0, 0)
24 ROJO_CLARO = (255, 0, 0)
25
26
27
28
29 # Constante para el evento de cambio de visibilidad de estrellas
30 CAMBIAR_VISIBILIDAD_ESTRELLAS_EVENTO = pygame.USEREVENT + 1
31
32 # Configura el temporizador para que genere el evento cada 1 segundo
33 pygame.time.set_timer(CAMBIAR_VISIBILIDAD_ESTRELLAS_EVENTO, 1000)
34
35 # Definir la cantidad de estrellas
36 CANTIDAD_ESTRELLAS = 100
37
38 # Función para crear el fondo estrellado
```



```
346 def pantalla_game_over():
347     while True:
348         for evento in pygame.event.get():
349             if evento.type == pygame.QUIT:
350                 pygame.quit()
351                 sys.exit()
352             elif evento.key == pygame.K_q:
353                 pygame.quit()
354                 sys.exit()
355             elif evento.type == CAMBIAR_VISIBILIDAD_ESTRELLAS_EVENTO:
356                 return False
357
358 def mostrar_ventana_nivel_superado():
359     ventana.fill(NEGRO)
360     draw_text(ventana, "¡Felicidades!!! Primer Nivel Superado!!!", 65, ANCHO / 2, ALTO / 4)
361
362     mouse = pygame.mouse.get_pos()
363     click = pygame.mouse.get_pressed()
364
365     boton_segundo_nivel = pygame.Rect(ANCHO / 2 - 100, ALTO / 2, 200, 50)
366     if boton_segundo_nivel.collidepoint(mouse):
367         pygame.draw.rect(ventana, VERDE_CLARO, boton_segundo_nivel)
368         if click[0] == 1:
369             # Ejecutar el archivo nivel2.py
370             os.system('python nivel2.py')
371             return False # Para terminar el bucle del primer nivel
372         else:
373             pygame.draw.rect(ventana, VERDE, boton_segundo_nivel)
374
375     draw_text(ventana, "Segundo nivel", 30, ANCHO / 2, ALTO / 2 + 10)
376     pygame.display.flip()
377
378 # Bucle para mantener la ventana emergente hasta que el usuario haga clic en el botón
379 ventana_abierta = True
380 while ventana_abierta:
381     for evento in pygame.event.get():
382         if evento.type == pygame.QUIT:
383             pygame.quit()
384             sys.exit()
385         if evento.type == pygame.MOUSEBUTTONDOWN:
386             if boton_segundo_nivel.collidepoint(evento.pos):
387                 ventana_abierta = False
```