

Introducción a la programación (CE1101)

Grupo 03

Profesor:

Leonardo Andrés Araya Martínez

Instituto Tecnológico Costa Rica

Escuela de Ingeniería en Computadores

Proyecto I Vintage Bomberman Game

Estudiante/Autor:

Anthony José Artavia Leitón

Documentación Externa

2024

Costa Rica

Tabla de contenido

Introducción	3
Objetivo general.....	3
Objetivos específicos	4
Descripción del Problema	4
Análisis de Resultados	5
Dificultades Encontradas	45
Bitácora de Actividades.....	50
Estadística de Tiempos	55
Conclusión	63
Bibliografía	64

Introducción

"Si puedes imaginarlo puedes programarlo" El mundo del desarrollo y la programación tienen una gran repercusión hoy en día, siendo así que múltiples áreas, tanto académicas como industrialmente productivas están estrechamente relacionadas con la creación de programas que faciliten al ciudadano promedio tener una mejor calidad de vida, simplificando tareas relativamente complicadas hacia una manera más fácil de comprender y usar.

Uno de los campos donde la programación tiene un impacto particularmente notable es el del entretenimiento, y más específicamente, el de los videojuegos. Este sector representa un vasto mercado en constante evolución, que implica la creación continua y progresiva de experiencias interactivas que cautiven y entretengan a los jugadores de todo el mundo.

En este contexto, el presente trabajo se enfoca en el desarrollo de un juego, explorando los procesos y desafíos involucrados en la creación de una experiencia de juego única y atractiva. A lo largo de estas páginas, examinaremos los fundamentos del diseño de juegos, la implementación de mecánicas de juego emocionantes y la importancia de una interfaz de usuario intuitiva para garantizar una experiencia de juego envolvente y gratificante.

A medida que avanzamos en nuestro análisis, descubriremos cómo la programación no solo es una herramienta para materializar ideas creativas, sino también un medio poderoso para inspirar, entretener y conectar a las personas a través del arte del juego.

Objetivo general

Desarrollar un videojuego mediante la aplicación de recursividad y la creación de interfaces gráficas.

Objetivos específicos

1. Elaborar una solución integral para abordar eficazmente el problema descrito en esta especificación, haciendo uso de técnicas de recursividad y habilidades especializadas en la manipulación de interfaces gráficas.
2. Desarrollar e implementar una solución eficiente utilizando los fundamentos de Python para resolver el problema detallado en esta especificación.
3. Elaborar la documentación correspondiente a la solución implementada para la evidencia del trabajo desarrollado utilizando estándares de documentación técnica y herramientas de gestión de proyectos

Descripción del Problema

En el marco de este proyecto, nos enfrentamos al desafío de diseñar e implementar el clásico juego 'Bomberman' en Python. Este juego requiere la integración de conceptos fundamentales de programación, como el manejo de datos básicos, la aplicación efectiva de la recursividad y la creación de una interfaz gráfica atractiva.

El objetivo central radica en desarrollar un sistema que permita a los jugadores explorar un laberinto, colocar estratégicamente bombas, evitar obstáculos y enemigos, todo mientras buscan alcanzar objetivos específicos. La complejidad del proyecto reside en la necesidad de utilizar la recursividad para abordar dinámicamente situaciones de juego, como la propagación de la explosión de las bombas y la gestión de eventos en tiempo real.

El componente de interfaz gráfica es esencial para brindar a los usuarios una experiencia visualmente atractiva y funcional. Esto implica la creación de una interfaz intuitiva que permita la interacción fluida con el juego, garantizando una experiencia de juego envolvente y entretenida.

Análisis de Resultados

Visualizaremos y analizaremos las diferentes ventanas de juego:

Menú principal:



```
import pygame

# Inicializar pygame
pygame.init()

# Definir dimensiones de la pantalla de inicio
W, H = 800, 600

#Nombre de la ventana
PANTALLA = pygame.display.set_mode((W, H))
pygame.display.set_caption("Menú Principal")

#ícono de la ventana
icono = pygame.image.load("Imágenes/Bombas/1.png")
pygame.display.set_icon(icono)

#Pantalla Información del creador
```

```

def informacion():
    import PantallaInformación
    PantallaInformación.mostrar_informacion_creador()

#Pantalla Configuración
def configuracion():
    import configuracion
    configuracion.configuracion()

#Iniciar juego
def iniciar_juego_bomberman():
    import Bomberman
    Bomberman.bucle_principal()

#Importamos el menú de mejores promedios:
def mejores_promedios_menu():
    import PantallaMejoresPuntajes
    PantallaMejoresPuntajes.bucle_principal()

# Paleta de colores para el fondo
Blanco = (255, 255, 255)
Negro = (0, 0, 0)

# Fuente de letra a usar
font = pygame.font.Font(None, 36)

# Creamos una función para dibujar texto en pantalla(Se le dan sus características como argumentos).
def dibujarTexto(text, font, color, surface, x, y):
    textobj = font.render(text, 1, color)
    textrect = textobj.get_rect()
    textrect.topleft = (x, y)
    surface.blit(textobj, textrect)

#Manejamos los posibles eventos que se pueden dar en esta ventana
def manejar_eventos(buttons):
    #Obtenemos la posición exacta del mouse(x,y), devuelve un a tupla con esos valores.
    mouse_posicion = pygame.mouse.get_pos()
    #Obtener el siguiente evento de la cola de eventos de Pygame.
    #Para manejar eventos uno por uno, devuelve un objeto de evento que tiene información del evento
    #más reciente.
    evento = pygame.event.poll()
    #Si el evento fue de cerrar la pestaña, cierra y finaliza todo.

```

```

    if evento.type == pygame.QUIT:
        pygame.quit()
        #Verifica si el tipo de evento fue de tipo click, si así lo fue llama la función
Botones_funciones(botones,mouse_Pos)
        #Esto con el fin de interactuar con el menú.

    if evento.type == pygame.MOUSEBUTTONDOWN:
        Botones_Funciones(buttons, mouse_posicion)

#Función para los botones, que harán si son precionados.
def Botones_Funciones(buttons, mouse_pos, indice=0):
    #Si el indice es menor a la cantidad de elementos en la lista de numeros entonces:
    if indice < len(buttons):
        #Definimos variables que serán el indice actual sobre el cual se está.
        boton, accion = buttons[indice]
        #Verificamos que el cursor del mouse está sobre el botón actual, Si es así, se ejecuta la
acción asociada al botón(action()), es decir se ejecuta cuando se le da click.
        if boton.collidepoint(mouse_pos):
            accion()
            #Si el cursor del mouse no está sobre el botón actual, se llama recursivamente a la
función "Botones_Funciones" con el siguiente indice, esto para pasar al siguiente botón
            #en la lista y repetir el proceso
        else:
            Botones_Funciones(buttons, mouse_pos, indice + 1)

#Boton salir del menú:
def salir_del_menu():
    pygame.display.update()
    pygame.quit()

def bucle_principal():

    ejecute = True
    while ejecute:

        #-----Sector de imagenes decorativas(inicio)-----
        # Cargar la imagen de fondo y escalarla a las dimensiones de la ventana
        fondo = pygame.image.load("Imagenes/Para el menu/Fondo.png")
        fondo = pygame.transform.scale(fondo, (W, H))
        # Dibujar la imagen de fondo en la pantalla
        PANTALLA.blit(fondo, (0, 0))

```

```

titulodelFondo = pygame.image.load("Imagenes/Para el menu/Titulo editado.jpg")
PANTALLA.blit(titulodelFondo,(100,50))

bomberDerecho = pygame.image.load("Imagenes/Para el menu/BomberDerecho.png")
bomberDerechoScale= pygame.transform.scale(bomberDerecho,(150,200))
PANTALLA.blit(bomberDerechoScale,(600,290))

bomberIzquierdo = pygame.image.load("Imagenes/Para el menu/bomberIzquierdo.png")
bomberIzquierdoScale = pygame.transform.scale(bomberIzquierdo,(150,200))
PANTALLA.blit(bomberIzquierdoScale,(80,290))

#-----Sector de imagenes decorativas(End)-----

# Creamos las variables de los rectángulos para poner texto en ellos luego
button_1 = pygame.Rect(275, 200, 250, 50)
button_2 = pygame.Rect(275, 300, 250, 50)
button_3 = pygame.Rect(275, 400, 250, 50)
button_4 = pygame.Rect(275, 500, 250, 50)
button_5 = pygame.Rect(100, 500, 160, 50)

# Utilizamos la función para dibujar los rectángulos con las especificaciones anteriormente
dadas
pygame.draw.rect(PANTALLA, Negro, button_1)
pygame.draw.rect(PANTALLA, Negro, button_2)
pygame.draw.rect(PANTALLA, Negro, button_3)
pygame.draw.rect(PANTALLA, Negro, button_4)
pygame.draw.rect(PANTALLA, Negro, button_5)

#Se me ocurrió la idea de que los botones tengan la apariencia de las cajas destructibles
BoxBoton = pygame.image.load("Imagenes/terrain/box.png")
BoxBotonScale = pygame.transform.scale(BoxBoton,(250,50))
BoxBotonScale2 = pygame.transform.scale(BoxBoton,(160,50))

PANTALLA.blit(BoxBotonScale,(275,200))
PANTALLA.blit(BoxBotonScale,(275,300))
PANTALLA.blit(BoxBotonScale,(275,400))
PANTALLA.blit(BoxBotonScale,(275,500))
PANTALLA.blit(BoxBotonScale2,(100,500))

# Utilizamos la función para dibujar texto creada al inicio
dibujarTexto("Iniciar Juego", font, Blanco, PANTALLA, 325, 210)
dibujarTexto("Configuración", font, Blanco, PANTALLA, 325, 310)
dibujarTexto("Mejores puntajes", font, Blanco, PANTALLA, 300, 410)

```



```

dibujarTexto("Salir", font, Blanco, PANTALLA, 370, 510)
dibujarTexto("Información", font, Blanco, PANTALLA, 110, 510)

# Lista con los botones y sus funciones (Lo que harán cuando sean presionados)
buttons = [
    (button_1, lambda: iniciar_juego_bomberman()),
    (button_2, lambda: configuracion()),
    (button_3, lambda: mejores_promedios_menu()),
    (button_4, lambda: salir_del_menu()),
    (button_5, lambda: informacion()),
]

# Si en manejar_eventos(buttons) se toma la decisión de cerrar el juego, entonces este bucle
principal while cesará
manejar_eventos(buttons)

# Actualizamos la pantalla
pygame.display.update()

# Ejecutar el menú principal
bucle_principal()

```

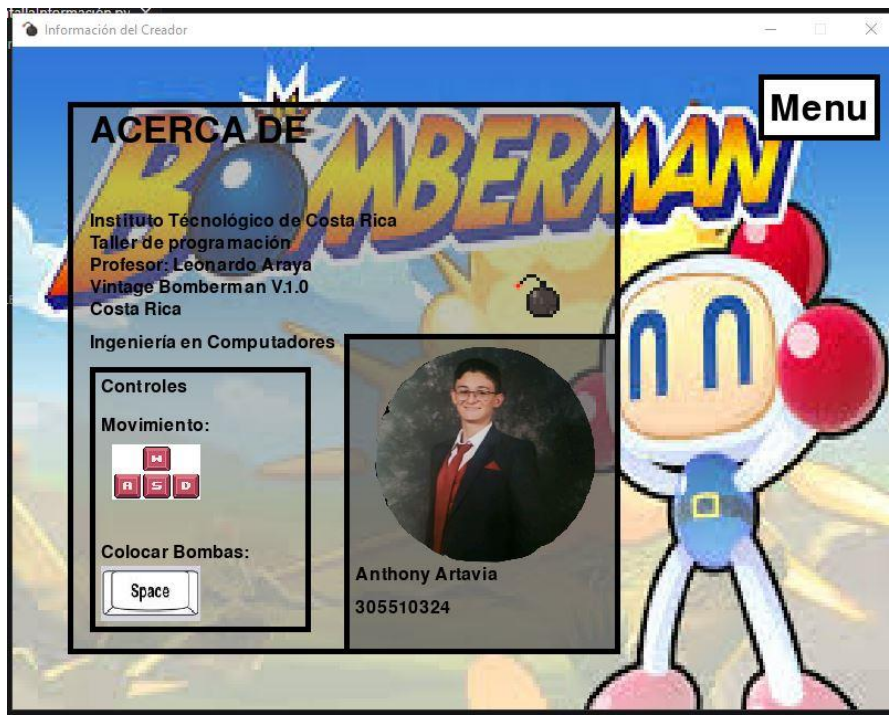
Funcionalidad:

En esta ventana se crea una interfaz atractiva para el usuario final, de manera que represente los aspectos básicos del programa, su funcionalidad principal se base en poder indexar/ importar los diferentes códigos de las otras ventanas de manera que al momento de hacer clic en un botón, este lleve a dicho lugar. Al igual que el bucle principal del juego, este se mantiene activo bajo el uso de recursividad y mediante un bucle while, en el cual se llaman las diferentes funciones encargadas de; Dibujar botones, Dibujar textos y las llamadas a los otros códigos. En este menú resalta la implementación de collidepoint para poder lograr detectar clicks en pantalla, el funcionamiento principal se base en dos funciones, manejar_eventos() y Botones_Funciones, primero en la función manejar_eventos se tiene la pila de eventos de pygame, la cual registra cada evento que se da, si un evento es de tipo “presionar botón del mouse” entonces le pasa a la función de ”Botones_Funciones” como argumento la posición del mouse obtenida con pygame.mouse.get_pos() y la lista de botones que se creó en el bucle principal.

Ya en la función de "Botones_Funciones()" se le pasa un argumento índice, en 0, y se hacen una serie de condicionales que determinan la acción y la posición del botón a presionar, esto se logra con el método de: `boton, accion = buttons[indice]` que según el índice del botón extrae que botón es y qué acción tiene asociada, si resulta que hay un click en esa posición, entonces se realiza "acción()", que corresponde a la acción asociada a ese botón la cual posteriormente se extrajo.

Finalmente el bucle se mantiene en constante actualización de pantalla con "pygame.display.update()" para lograr hacer que los elementos se mantengan dibujados y no queden atrás en la pila de llamadas.

Pantalla de Información (Acerca de):



```
import pygame

# Inicializar Pygame
pygame.init()

# Configuración de la ventana
PANTALLA_ANCHO= 800
PANTALLA_ALTO = 600
PANTALLA = pygame.display.set_mode((PANTALLA_ANCHO, PANTALLA_ALTO))
```

```

pygame.display.set_caption("Información del Creador")

#ícono de la ventana
icono = pygame.image.load("Imágenes/Bombas/1.png")
pygame.display.set_icon(icono)

# Colores
Blanco = (255, 255, 255)
Negro = (0, 0, 0)

#Fuentes a usar
font_titulo = pygame.font.Font(None, 48)
font_texto = pygame.font.Font(None, 24)

# Color grisáceo con transparencia ajustable
GrisTransparente = (128, 128, 128, 128) # Ajusta el último valor (alfa) para cambiar la
transparencia

# Cargar imagen de fondo
imagen_fondo = pygame.image.load("Imágenes/Info creador/Fondo.jpeg")
imagen_fondo = pygame.transform.scale(imagen_fondo, (PANTALLA_ANCHO, PANTALLA_ALTO)) # Ajustar
tamaño de la imagen

# Cargar imagen del creador
foto = pygame.image.load("Imágenes/Info creador/Creador.png")
foto = pygame.transform.scale(foto, (200, 200)) # Ajustar tamaño de la foto

#Cargamos la imagen de las teclas a usar
fotoMovimiento = pygame.image.load("Imágenes/Info creador/Movimiento.png")
fotoMovimiento = pygame.transform.scale(fotoMovimiento, (80, 50)) # Ajustar tamaño de la foto

#Cargamos la imagen de la tecla espacio para las bombas
fotoBombasEspacio = pygame.image.load("Imágenes/Info creador/Bombas.JPG")
fotoBombasEspacio = pygame.transform.scale(fotoBombasEspacio, (90, 50)) # Ajustar tamaño de la foto

#Cargamos una imagen de las bombas para decoración
fotoBombasDefondo = pygame.image.load("Imágenes/Bombas/1.png")
fotoBombasDefondo = pygame.transform.scale(fotoBombasDefondo, (50, 50)) # Ajustar tamaño de la foto

# Función para mostrar texto en pantalla(Reutilizable)
def draw_text(text, font, color, x, y):
    text_surface = font.render(text, True, color)
    text_rect = text_surface.get_rect()

```

```

    text_rect.topleft = (x, y)
    PANTALLA.blit(text_surface, text_rect)

# Función para dibujar los botones en la pantalla de información del creador
def dibujar_botones():
    pygame.draw.rect(PANTALLA, Negro, (675, 25, 110, 60), 5)
    VolverAlMenu = pygame.Rect(680, 30, 100, 50)
    pygame.draw.rect(PANTALLA, Blanco, VolverAlMenu)
    draw_text("Menu", font_titulo, Negro, 685, 40)

# Función principal
ejecuta = True #-----> Básicamente lo que enciende esta pantalla
def mostrar_informacion_creador():
    global font_texto, font_titulo
    # Bucle principal
    while ejecuta:

        # Dibujar imagen de fondo
        PANTALLA.blit(imagen_fondo, (0, 0))

        # Dibujar rectángulo transparente en el área del marco de información
        Fondo_Gris = pygame.Surface((500, 500), pygame.SRCALPHA) # Superficie transparente
        Fondo_Gris.fill((128, 128, 128, 128)) # Color grisáceo transparente
        PANTALLA.blit(Fondo_Gris, (50, 50))

        # Dibujar rectángulo transparente en el área del marco de autor
        Fondo_Gris = pygame.Surface((250, 290), pygame.SRCALPHA) # Superficie transparente
        Fondo_Gris.fill((64, 64, 64, 128)) # Color grisáceo transparente
        PANTALLA.blit(Fondo_Gris, (300, 260))

        # Dibujar marco de información
        pygame.draw.rect(PANTALLA, Negro, (50, 50, 500, 500), 5)
        # Marco para la foto
        pygame.draw.rect(PANTALLA, Negro, (300, 260, 250, 290), 5)
        # Marco para la info de controles
        pygame.draw.rect(PANTALLA, Negro, (70, 290, 200, 240), 5)

        # Dibujar texto de información
        draw_text("ACERCA DE", font_titulo, Negro, 70, 60)
        draw_text("Anthony Artavia", font_texto, Negro, 310, 470)
        draw_text("305510324", font_texto, Negro, 310, 500)
        draw_text("Instituto Tecnológico de Costa Rica", font_texto, Negro, 70, 150)
        draw_text("Taller de programación", font_texto, Negro, 70, 170)

```

```

draw_text("Profesor: Leonardo Araya", font_texto, Negro, 70, 190)
draw_text("Vintage Bomberman V.1.0", font_texto, Negro, 70, 210)
draw_text("Costa Rica", font_texto, Negro, 70, 230)
draw_text("Ingeniería en Computadores", font_texto, Negro, 70, 260)
draw_text("Controles", font_texto, Negro, 80, 300)
draw_text("Movimiento:", font_texto, Negro, 80, 335)
draw_text("Colocar Bombas:", font_texto, Negro, 80, 450)

#Dibujar foto de las teclas
PANTALLA.blit(fotoMovimiento,(90,360))
# Dibujar foto del creador
PANTALLA.blit(foto, (327, 270))
#Dibujar tecla espacio
PANTALLA.blit(fotoBombasEspacio, (80, 470))
#Dibujar bombas de decoración
PANTALLA.blit(fotoBombasDefondo,(450,200))

#Dibujamos la función que diuja botones, en este caso solo el de volver, pero se puede
dibujar más solo llamandolos acá
dibujar_botones()

#Si se preciona la X de la ventana, cerramos la pantalla
event = pygame.event.poll()
if event.type == pygame.QUIT:
    pygame.quit()

    #Lógica para hacer click en el botón y devuelva al menú principal
elif event.type == pygame.MOUSEBUTTONDOWN:
    mouse_pos = pygame.mouse.get_pos()
    if 675 <= mouse_pos[0] <= 785 and 25 <= mouse_pos[1] <= 85: # Verificar si se hizo clic
en el botón de volver al menú principal
        return # Salir de la función y volver al menú principal

# Actualizar pantalla
pygame.display.update()

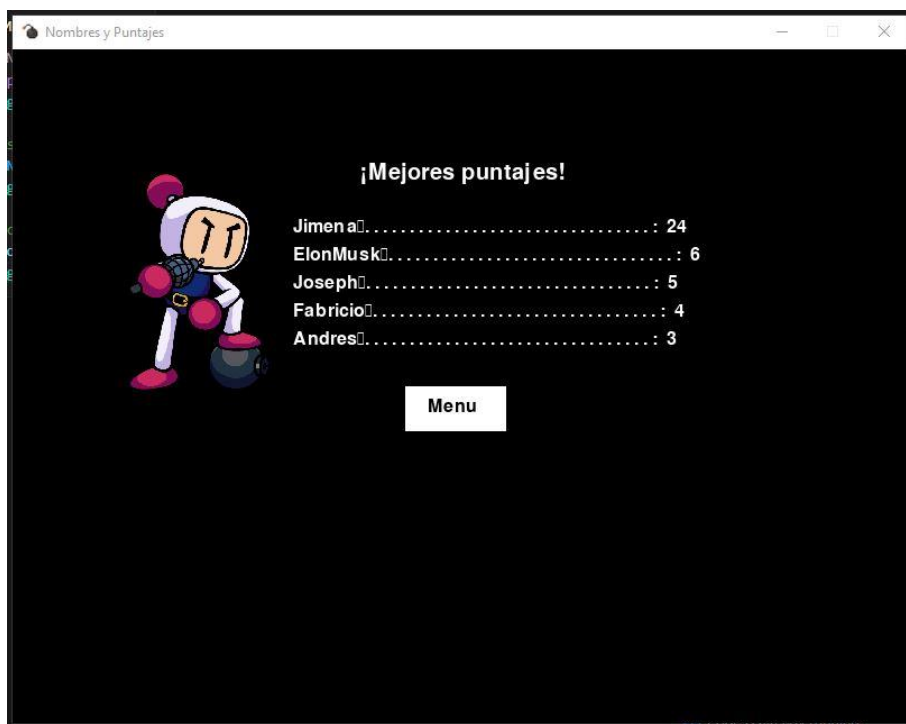
# Ejecutar la ventana de información del creador
mostrar_informacion_creador()

```

Funcionalidad:

En esta pantalla se implementó el uso de rectángulos para dibujar sectores de la pantalla como marcos y espacios separados, como en este menú solo hay un botón y es el de volver al menú, se utilizó un nuevo método para la detección de clicks, y es que en la pila de eventos si se detecta un evento de tipo clic, entonces si ese evento de tipo click está entre las coordenadas del botón, hace un return, para poder salir de la función y volver al menú principal.

Menú mejores puntajes:



```
import pygame
pygame.init()

#Establecemos las proporciones de la ventana y algunas características
PANTALLA = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Nombres y Puntajes")

#ícono de la ventana
icono = pygame.image.load("Imagenes/Bombas/1.png")
pygame.display.set_icon(icono)
```

```

#Paleta de colores
blanco = (255,255,255)
Negro = (0, 0, 0)

font_texto = pygame.font.Font(None, 24)
font_titulo = pygame.font.Font(None,30)

#Al final descubrí que pygame no tiene soporte nativo para mover las imagenes de los gifs, pero me
gustó la imagen
#entonces lo dejo así.
gif = pygame.image.load("Imagenes/Imagenes mejores puntajes/BombermanGif.gif")
gifScalado = pygame.transform.scale(gif,(135,205))

# Función para leer nombres desde archivo de texto
def leer_nombres(txtConLosNombres):
    #El contenido de la primera línea que lee se almacena en reglon
    reglon = txtConLosNombres.readline()
    if not reglon: #Caso base, si no hay reglón, entonces llegamos al final
        return [] #Ponemos un vacío para que no afecte al resto
    return [reglon] + leer_nombres(txtConLosNombres)

# Función para leer puntajes desde archivo de texto
def leer_puntajes(txtConPuntuaciones):
    #El contenido de la primera línea que lee se almacena en reglon
    reglon = txtConPuntuaciones.readline()
    if not reglon: #Caso base, si no hay reglón, entonces llegamos al final
        return [] #Igual, ponemos un vacío para no afectar a la salida
    return [int(reglon)] + leer_puntajes(txtConPuntuaciones)

def mostrar_mejores_puntajes(PANTALLA, fuenteParaTexto, lista_combinada, indice=0): #Definir el
indice como 0 y como argumento al mismo tiempo lo vimos en clase
    #muy útil para no tener que ingresar otro valor,
es decir no nos preocupamos por poner otro valor
    if indice >= min(5, len(lista_combinada)): #Así limitamos la especificación de "Que sean los
mejores 5 promedios" haciendo que si lee más de 5
        #entonces la función simplemente sale de la ejecución y
no devuelve nada explícitamente.
        return#Si la lista tiene menos de cinco elementos, simplemente usará la longitud de la lista
como límite.

    # Ordenamos la lista combinada por puntaje de mayor a menor

```

```

    lista_combinada.sort(key=lambda x: x[1], reverse=True)#El argumento de lambda define que el
ordenamiento se hará basado en el segundo elemento de cada tupla(los puntajes)
    #reverse true es para que se de manera descendente

    nombre, puntaje = lista_combinada[indice]
    text_surface = fuenteParaTexto.render(f"{nombre}. . . . .")
    . . . . . : {puntaje}", True, blanco)
    PANTALLA.blit(text_surface, (250, 150 + indice * 25))

    mostrar_mejores_puntajes(PANTALLA, fuenteParaTexto, lista_combinada, indice + 1)

# Función para mostrar texto en pantalla(Reutilizable)
def draw_text(text, font, color, x, y):
    text_surface = font.render(text, True, color)
    text_rect = text_surface.get_rect()
    text_rect.topleft = (x, y)
    PANTALLA.blit(text_surface, text_rect)

# Función para dibujar los botones en la pantalla de información del creador
def dibujar_botones():
    VolverAlMenu = pygame.Rect(350, 300, 90, 40)
    pygame.draw.rect(PANTALLA, blanco, VolverAlMenu)
    draw_text("Menu", font_texto, Negro, 370, 310)

ejecuta = True
def bucle_principal():
    global ejecuta
    if ejecuta == False:
        pygame.quit()
        return

    PANTALLA.fill(Negro)

    # Fuente o fuentes a usar
    fuenteParaTexto = pygame.font.Font(None, 24)
    draw_text("¡Mejores puntajes!", font_titulo, blanco, 310, 100)

    # Leer los nombres
    with open('nombres.txt', 'r') as txtConLosNombres:
        nombres = leer_nombres(txtConLosNombres)

    # Leer los puntajes

```



```

with open('puntajes.txt', 'r') as txtConPuntuaciones:
    puntajes = leer_puntajes(txtConPuntuaciones)

# Combinar nombres y puntajes en una lista de tuplas
lista_combinada = list(zip(nombres, puntajes))

mostrar_mejores_puntajes(PANTALLA, fuenteParaTexto, lista_combinada)
dibujar_botones()

while ejecuta:
    event = pygame.event.poll()

    if event.type == pygame.QUIT:
        ejecuta = False
    elif event.type == pygame.MOUSEBUTTONDOWN:
        mouse_pos = pygame.mouse.get_pos()
        if 350 <= mouse_pos[0] <= 440 and 300 <= mouse_pos[1] <= 340:
            print("Mensaje para probar la funcionalidad del botón volver")
            return

    # Dibujar la imagen GIF en la pantalla
    PANTALLA.blit(gifScalado, (100, 100))
    pygame.display.update()

pygame.quit()

bucle_principal()

```

Funcionalidad:

En esta pantalla se logró implementar una funcionalidad nunca experimentada, y fue adquirir el conocimiento de que Python pueden crear, leer y escribir documentos de texto(txt), esto se utilizará para almacenar los nombres de las personas jugadores y su respectiva puntuación, esto anterior se explica mejor en el apartado principal del código del juego, en esta pantalla lo que se hace es abrir los documentos de texto y se le da la posibilidad de solo lectura, ya que no es interesa que solamente se muestre por pantalla su contenido, se utilizó de esta manera:

```

# Leer los nombres
with open('nombres.txt', 'r') as txtConLosNombres:
    nombres = leer_nombres(txtConLosNombres)

```

De igual forma para los puntajes, mi resolución fue la de crear un txt para almacenar los nombres y otro para los puntajes, entonces la opción de Python logra abrir dicho documento y se le otorga su contenido como una variable y se le dice en otra función que, por reglón vaya leyendo los nombres y va sumando a la lista retornable hasta que ya no quedan más reglones que analizar. Posteriormente los nombres de jugadores, se almacenan en la variable nombres, este proceso se hace también con los puntajes, posteriormente se ingresan a:

```
# Combinar nombres y puntajes en una lista de tuplas
lista_combinada = list(zip(nombres, puntajes))
```

Donde con la funcionalidad zip, agrupa ambas listas y las funciona en una lista de tuplas, que luego son llevadas mediante esa nueva variable a la función encargada de mostrar los mejores puntajes por pantalla. A resumidas cuentas lo que se hace dentro de la función es limitar la cantidad de lecturas a 5, para poder mostrar solo los 5 mejores, este es el caso base, si se alcanzan 5 índices de la lista de tuplas, entonces hace return.

Con la palabra reservada sort, se ordena a la variable que tiene almacenadas las tuplas y se le dice, haga el sort con el segundo elemento de cada tupla, ósea los puntajes y además haga reverse, para poder mostrar por pantalla los puntajes de forma descendentes.

Luego simplemente se renderiza llamando a las funciones que dibujan texto y pasándoles las variables nombre y puntuación. Este proceso se hace de forma recursiva.

Menú de configuración (Música):



```
import pygame
import sys

# Inicializar Pygame
pygame.init()

# Configuración de la pantalla
W = 800
H = 600
screen = pygame.display.set_mode((W, H))
pygame.display.set_caption("Configuración de Juego")
#ícono de la ventana
icono = pygame.image.load("Imagenes/Bombas/1.png")
pygame.display.set_icon(icono)

# Colores
Blanco = (255, 255, 255)
Negro = (0, 0, 0)

# Función para mostrar texto en pantalla
def draw_text(text, font, color, x, y):
    text_surface = font.render(text, True, color)
    text_rect = text_surface.get_rect()
    text_rect.center = (x, y)
    screen.blit(text_surface, text_rect)

# Función para cargar música
def cargar_musica():
    pygame.mixer.music.load("Sound/Bomberman music.mp3")
    pygame.mixer.music.play(-1) # -1 hace que la música se repita indefinidamente

# Función para detener la música
def detener_musica():
    pygame.mixer.music.stop()

# Función principal
def configuracion():
    # Fuentes
    font = pygame.font.Font(None, 36)

    # Estado de la música (habilitada o deshabilitada)
    musica_habilitada = True
```

```

def dibujarConfiguracion():
    fondo = pygame.image.load("Imagenes/Menu de configuración fondo/Fondo.JPG")
    fondo = pygame.transform.scale(fondo, (W, H))
    # Dibujar la imagen de fondo en la pantalla
    screen.blit(fondo, (0, 0))
    draw_text("Configuración de Juego", font, Negro, W // 2, 50)

    Bomber = pygame.image.load("Imagenes/Menu de configuración fondo/Music Bomberman.png")
    BomberScale = pygame.transform.scale(Bomber, (200, 200))
    screen.blit(BomberScale, (500, 200))

    # Dibujar botones de música
    if musica_habilitada:
        draw_text("Música: Habilitada", font, Negro, W // 2, 200)
    else:
        draw_text("Música: Deshabilitada", font, Negro, W // 2, 200)

    # Dibujar botón de volver
    draw_text("Volver", font, Negro, W // 2, 400)

    # Actualizar la pantalla
    pygame.display.update()

dibujarConfiguracion() # Dibujar pantalla inicial

# Bucle para manejar eventos
ejecuta = True
while ejecuta:
    event = pygame.event.poll()
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    elif event.type == pygame.MOUSEBUTTONDOWN:
        mouse_pos = pygame.mouse.get_pos()
        if 350 <= mouse_pos[0] <= 450 and 180 <= mouse_pos[1] <= 220:
            if musica_habilitada:
                detener_musica()
                musica_habilitada = False
            else:
                cargar_musica()
                musica_habilitada = True
        dibujarConfiguracion() # Redibujar pantalla con cambios
    elif 350 <= mouse_pos[0] <= 450 and 380 <= mouse_pos[1] <= 420:

```

```
        return # Volver al menú principal

# Ejecutar la configuración
configuracion()
```

Funcionalidad:

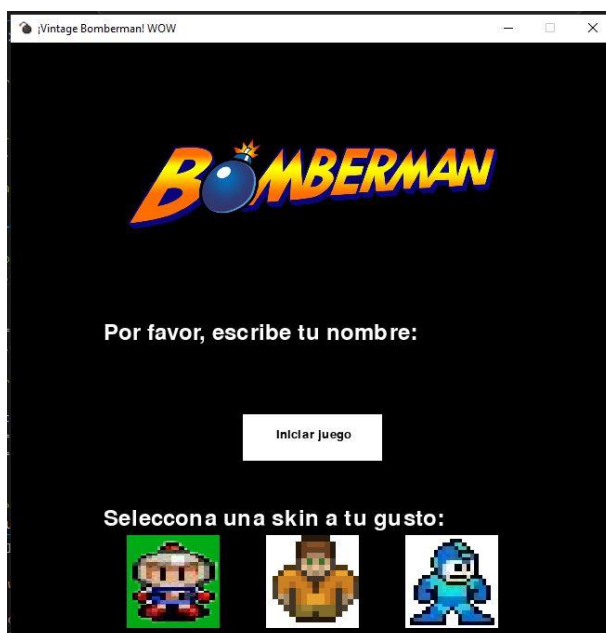
En este menú se destaca la opción de poder desactivar y activar música, realmente no es tan complejo, primero se carga la música con la función de `pygame.mixer.music.load()`, luego se le dice `pygame.mixer.music.play(-1)` para que la música de forma infinita, luego en manejar evento, si presiona sobre las coordenadas del botón que activa o desactiva la música, depende, si `música_habilitada` que por defecto está en `True`, entonces se llama a una función que detiene la música, se hace con:

```
pygame.mixer.music.stop()
```

Si fuera el caso contrario, entonces se vuelve a cargar la música, si se presiona otro lugar que no sea las del botón de música (en este caso las otras coordenadas) se hace `return` y se vuelve al menú.

Juego principal:

Primero aparece un submenú post-juego, donde la persona puede ingresar su nombre y seleccionar una skin.



```
import pygame

# Inicialización de Pygame
pygame.init()

# Pantalla - Ventana (Proporciones)
W, H = 650, 650
PANTALLA = pygame.display.set_mode((W,H))

#Título
pygame.display.set_caption("¡Vintage Bomberman! WOW")

#ícono de la ventana
icono = pygame.image.load("Imágenes/Bombas/1.png")
pygame.display.set_icon(icono)

#Sector de paleta de colores
gris = (146, 143, 136)
grisOscuro = (118, 109, 94)
negro = (0,0,0)
verde = (0, 171, 21)
blanco = (255,255,255)

# Variable para almacenar el nombre del jugador
nombre_jugador = ""

#Nivel inicial
nivel = 1

# Variable para almacenar la skin seleccionada (inicialmente ninguna)
skin_seleccionada = None

#Dibujar logo
imagen_logo = pygame.image.load("Imágenes/Logo/logo.png")
imagen_logo = pygame.transform.scale(imagen_logo, (400, 100))

#------(Inicio)Cargar imagenes para los botones skins-----

imagen_skin1 = pygame.image.load("Imágenes/Player/Down/Down1.png")
imagen_skin1 = pygame.transform.scale(imagen_skin1, (100, 100))

imagen_skin2 = pygame.image.load("Imágenes/Player2/Down/1.png")
```

```

imagen_skin2 = pygame.transform.scale(imagen_skin2, (100, 100))

imagen_skin3 = pygame.image.load("Imagenes/Player3/quieto.png")
imagen_skin3 = pygame.transform.scale(imagen_skin3, (100, 100))

#------(End)Cargar imagenes para los botones skins-----

# Función para renderizar texto en la pantalla
def renderizar_texto(texto, longitud, x, y, color):
    font = pygame.font.Font(None, longitud)
    superficie_texto = font.render(texto, True, color)
    PANTALLA.blit(superficie_texto, (x, y))

# Función para dibujar un botón
def dibujar_boton(texto, x, y, ancho, alto, color_activo, color_inactivo):
    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()

    if x + ancho > mouse[0] > x and y + alto > mouse[1] > y:
        pygame.draw.rect(PANTALLA, color_activo, (x, y, ancho, alto))
        if click[0] == 1:
            return True
    else:
        pygame.draw.rect(PANTALLA, color_inactivo, (x, y, ancho, alto))

    renderizar_texto(texto, 20, x + ancho / 2 - len(texto) * 3, y + alto / 2 - 10, negro)

    return False

# Función para guardar el nombre del jugador en un archivo de texto #Aprendí esto en una consulta
con el profe Leonardo
def guardar_texto(nombre_jugador):
    #la "a" es de append. , es para que agregue nombres uno tras otro y no borre o sobre escriba
    with open("nombres.txt", "a") as archivo:
        archivo.write(nombre_jugador + "\n")

# Bucle para la ventana inicial de ingreso de nombre
# Función recursiva para la ventana inicial de ingreso de nombre y selección de skin
def manejar_eventoMenu(ventana_activa=True):
    global nombre_jugador
    global skin_seleccionada

    while ventana_activa:

```

```

event = pygame.event.poll() # Para hacer que los eventos vengan en cola
if event.type == pygame.QUIT:
    pygame.quit()
    return
elif event.type == pygame.KEYDOWN:
    if event.key == pygame.K_BACKSPACE:
        nombre_jugador = nombre_jugador[:-1]
    else:
        nombre_jugador += event.unicode

# Color de la pantalla antes del juego
PANTALLA.fill(negro)

#Dibujar logo
PANTALLA.blit(imagen_logo,(125,100))

# Renderizar texto para ingresar el nombre
renderizar_texto("Por favor, escribe tu nombre:", 36, 100, 300, blanco)
renderizar_texto(nombre_jugador, 36, 100, 350, blanco)

renderizar_texto("Seleccona una skin a tu gusto: ",35,100,500,blanco)
# Dibujar botón de inicio
if dibujar_boton("Iniciar juego", 250, 400, 150, 50, verde, blanco):
    #Cuando el jugador le da al botón iniciar juego, esta ventana se cierra pero guarda su
    nombre en un archivo de texto(Aprendí de esto con una consulta del profe Leonardo)
    guardar_texto(nombre_jugador)
    ventana_activa = False
    #Si la persona le da a iniciar juego, entonces la ventana se pone en False, lo que
    finaliza el bucle de esta ventana y la cierra
    #para poder continuar con el juego(Solo si está activa: if ventana_activa:
"manejar_evento(ventana_activa)")

# Dibujar botones para seleccionar skin
if dibujar_boton("Skin 1", 125, 530, 100, 100, verde if skin_seleccionada == 1 else blanco,
verde if skin_seleccionada == 1 else blanco):
    skin_seleccionada = 1
if dibujar_boton("Skin 2", 275, 530, 100, 100, verde if skin_seleccionada == 2 else blanco,
verde if skin_seleccionada == 2 else blanco):
    skin_seleccionada = 2
if dibujar_boton("Skin 3", 425, 530, 100, 100, verde if skin_seleccionada == 3 else blanco,
verde if skin_seleccionada == 3 else blanco):
    skin_seleccionada = 3

```



```

#Dibujar skins sobre los botones:
PANTALLA.blit(imagen_skin1,(125,530))
PANTALLA.blit(imagen_skin2,(275,530))
PANTALLA.blit(imagen_skin3,(425,530))
# Si la persona no seleccionó ninguna skin, por defecto se le asigna la 1
if skin_seleccionada is None:
    skin_seleccionada = 1

# Actualizar la pantalla
pygame.display.update()

# Llamada inicial a la función recursiva
manejar_eventoMenu()

```

Este código va de primero en el script principal del juego, de manera que pueda aparecer en pantalla antes que el resto del código, su función es la de presentar la introducción al juego, pero antes debe de seleccionar una skin y opcional ingresar un nombre, la selección de skin tiene un funcionamiento bastante interesante, en esta sección se explicará solo la parte correspondiente a este submenú el resto pertenece al código del juego como tal.

Cada botón de skin es en realidad un cuadrado el cual se le transpuso la imagen correspondiente asociada a la skin, en este menú se implementó lo aprendido en el resto de ventanas para poder aplicar colisión con el clic del mouse, vamos a explicar con un ejemplo; por defecto la skin está seteada en la #1 esto quiere decir que siempre cuando se inicie el juego, va a ingresar con la skin #1, a menos que la persona la cambie, veamos ese caso: La persona decide seleccionar la #2, entonces los condicionales dice que si se selecciona el botón 2, entonces, se llama a la función dibujar_boton() la cual tiene no solo como funcionalidad dibujar los botones si no detectar los clics según las coordenadas de los botones en lista, entonces si se detecta dicha colisión, entonces el fondo del botón pasa a color verde y la variable de skin seleccionada pasa a ser 2.

Realmente esta es la función relevante de esta ventana, otra funcionalidad que tiene es almacenar los nombres que se escriben en el documento de texto que anteriormente se mencionó, dicho txt es leído por la ventana de mejores puntajes, ahí se explica que pasa con los nombres almacenados.

Código principal del juego:



Como el código es extenso, se explicarán las funciones más relevantes del juego:

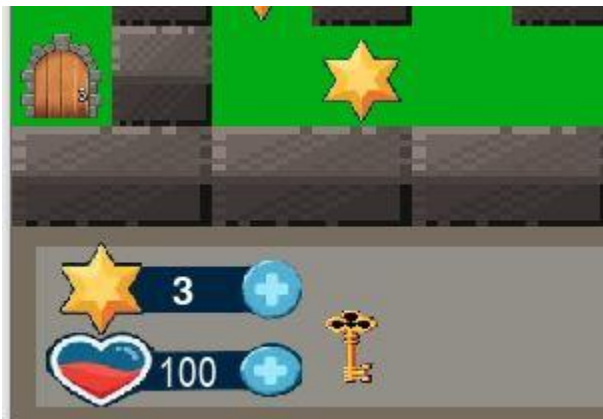
Llave escondida y puerta para la progresión:

- **Llave:**

La llave se encuentra dentro de los bloques destructibles, esto se logró dibujando primero la llave en el buble principal y luego los bloques destructibles para dar la sensación de que estaba dentro de los bloques:



La llave se puede recoger gracias a que está dibujada sobre un rectángulo, de hecho se implementó la misma funcionalidad que en las pantallas pasadas pero esta vez la colisión se da con las coordenadas del jugador, entonces si el jugador colisiona con la llave, la llave se deja de dibujar pero realmente se transpone el rectángulo de color verde que tiene atrás, entonces prácticamente el rectángulo verde la tapa y se la ilusión óptica de que se recogió, pero realmente está detrás, la forma en que podemos ver como se dibuja la llave en el inventario, es porque si la colisión se da, entonces la llave se transpone atrás y luego se dibuja otra imagen de la llave pero en otra coordenada, es decir en las del inventario:



- **Puerta:**

El principio es similar al de la llave solo que en este caso si la llave fue recogida, entonces permite que se dé la colisión con la puerta, si esta se da, entonces el nivel mundo sube en 1, haciendo que aparezca un mensaje en pantalla que indica que se ha subido de nivel, se hace un delay para que de la ilusión de progresión y luego otros condicionales en el bucle principal permiten que si es nivel 2 o el que sea, se agregen funciones que dibujen las diferentes puertas y llaves en sus determinadas coordenadas.

En resumen, las funciones de la llave y la puerta realmente son iguales en todos los niveles lo que cambia es la apariencia de la llave y además donde tienen permitido dibujarse y cuando, en el bucle principal, como se mencionó, dependiendo del nivel del mundo, permite o no a las funciones dibujarse:

```
#Como lo que se llama primero se dibuja primero, ll
#tapen la llave
if nivel == 1:
    PANTALLA.blit(puerta_img, puerta_rect)
    detectar_colisionLlavePuertaNivel1(px,py)
    renderizar_elementosNivel1()

if nivel == 2:
    PANTALLA.blit(puerta2_img, puerta2_rect)
    detectar_colisionPuertaNivel2(px, py)
    renderizar_elementosPuertaNivel2()

if nivel == 3:
    PANTALLA.blit(puerta3_img, puerta3_rect)
    detectar_colisionPuertaNivel3(px, py)
    renderizar_elementosPuertaNivel3()
```

Movimiento y Skins:

Anteriormente como ejemplo se indicó que la skin seleccionada era la #2, entonces en el diccionario de skins, la 2 corresponde a una serie de imágenes que se llaman sucesivamente para dar la animación correspondiente a una caminata, la función encargada de recibir que skin se seleccionó es `MovimientopersonajeJugable()`, la cual dice, la skin seleccionada ahora es “skin” y hace una serie de aumentos en los índices del diccionario para ir cambiando las imágenes según sea el movimiento(arriba, abajo, izquierda, derecha). Es importante aclarar que las variables de movimiento de las 4 direcciones y las de cuenta pasos, por defecto están en False, ya que, si no da error, y el personaje camina solo o las imágenes se transponen de manera errónea. Esta función recibe como parámetro cualquier skin seleccionada del diccionario “skins”.

- **Colisión con los bloques destructibles e indestructibles:**

Dentro del bucle principal del juego se implementó la mecánica de que si se presionan determinadas teclas en este caso (A,D,W,S) entonces se hacen comprobaciones para determinan la dirección a la cual se moverá el personaje:

```
keys = pygame.key.get_pressed()
# Tecla W - Movimiento hacia arriba
if keys[pygame.K_a] and px > velocidad and not check_collision(px - velocidad, py,
muros_Indestructibles_posiciones, muros_destructibles_posiciones):
    px -= velocidad
    izquierda = True
    derecha = False
    arriba = False
    abajo = False
elif keys[pygame.K_d] and px < W - 32 - velocidad and not check_collision(px + velocidad,
py, muros_Indestructibles_posiciones, muros_destructibles_posiciones):
    px += velocidad
    derecha = True
    izquierda = False
    arriba = False
    abajo = False
elif keys[pygame.K_w] and py-30 > velocidad and not check_collision(px, py - velocidad,
muros_Indestructibles_posiciones, muros_destructibles_posiciones):
    py -= velocidad
    arriba = True
    abajo = False
    derecha = False
    izquierda = False
elif keys[pygame.K_s] and py < 571 - 114 and not check_collision(px, py + velocidad,
muros_Indestructibles_posiciones, muros_destructibles_posiciones):
    py += velocidad
    abajo = True
    arriba = False
    derecha = False
    izquierda = False
else: #Si no se preciona nada, todo en Falso para que el personaje no se mueva solo
    arriba = False
    abajo = False
    derecha = False
    izquierda = False
```

Pero lo importante aquí está en la función `check_collision()` que se llama en cada línea, esta dice si no se da “La función `check_collision()`” permita moverse, esta función tiene como finalidad determinar si hay colisión con los bloques destructibles e indestructibles(se explica más adelante que son y como funcionan).

```
#Creamos una función recursiva que controle las colisiones, toma las coordenadas X y Y además la lista de coordenadas de los bloques,
def check_collision(x, y, muros_Indestructibles_posiciones, muros_destructibles_posiciones):
    if not muros_Indestructibles_posiciones and not muros_destructibles_posiciones:
        # Si no quedan muros ni bloques destructibles por verificar, retornar False
        return False

    if muros_Indestructibles_posiciones:
        # Verificar colisión con muros
        muro_x, muro_y = muros_Indestructibles_posiciones[0]
        if x < muro_x + 46 and x + 32 > muro_x and y < muro_y + 20:
            return True

    if muros_destructibles_posiciones: #Se agregó esta parte cuando se incluyeron los bloques que si se pueden destruir
        # Verificar colisión con bloques destructibles
        muro_x, muro_y = muros_destructibles_posiciones[0]
        if x < muro_x + 46 and x + 32 > muro_x and y < muro_y + 20:
            return True

    # Llamada recursiva para comprobar la colisión con el resto de muros y bloques destructibles
    return check_collision(x, y, muros_Indestructibles_posiciones[1:], muros_destructibles_posiciones[1:])

#------(Start)(Función que dibuja la Pantalla Final)-----
```

A resumidas cuentas la función `check_collision()`, recibe como parámetros los argumentos de, posición en X, la posición en Y, la lista de muros indestructibles_posiciones, seguido dice, si hay muros ya sea destructibles o indestructibles, entonces extraiga las coordenadas del bloque desde la lista, y pregunte, si la coordenadas actuales del personaje ajustadas para verificar la coordenada actual del bloque y la posición actual del bloque colisionan, si se da, entonces retorne True caso contrario False, lo que limita el movimiento al personaje principal al no poder moverse a más de esas coordenadas.

Bloques Destructibles e indestructibles:

Esta sección se explica cómo funcionan los bloques destructibles e indestructibles, se tienen 4 funciones principales para el dibujo de los muros:

- **`draw_bordes_y_temporizador()`:**

Esta es la función principal que dibujará los muros superiores e inferiores además de los bordes donde se situará la vida, bombas disponibles, puntos y demás...

```
#Función principal
def draw_bordes_y_temporizador():
    pygame.draw.rect(PANTALLA, grisOscuro, (0, 550, 800, 90)) # Borde inferior
    pygame.draw.rect(PANTALLA, gris, (12, 560, 623, 90)) # Área de juego
    pygame.draw.rect(PANTALLA, grisOscuro, (0, 640, 800, 90)) # Borde inferior

    # Cargar la imagen del muro y ajustar su tamaño
    muro_imagen = pygame.image.load("Imagenes/terrain/block.png")
    muro_imagen = pygame.transform.scale(muro_imagen, (93, 50)) # Ajustar tamaño del muro

    # Definir la cantidad de muros consecutivos a dibujar en los bordes superior e inferior
    cantidad_muros_superiores = 7
    cantidad_muros_inferiores = 7

    # Dibujar varios muros consecutivos a lo largo del borde superior de forma recursiva
    draw_muros_superiores_aux(muro_imagen, cantidad_muros_superiores)

    # Dibujar varios muros consecutivos a lo largo del borde inferior de forma recursiva
    draw_muros_inferiores_aux(muro_imagen, cantidad_muros_inferiores)
```

Lo relevante de esta función es que llama a dos auxiliares encargadas de dibujar los bordes superiores e inferiores, esto se logra primero manualmente según las dimensiones de la ventana de juego, sacar la cantidad de bloques que cabrán horizontalmente de manera que parezcan un muro, se multiplica el ancho de cada bloque por la cantidad de bloques, la cantidad será de acuerdo a los que puedan rellenar de izquierda a derecha la pantalla, en este caso si cada bloque es de un largo de 93 pixeles, 7 seguidos dará en total 651 pixeles, se pasa por 1 a la longitud de la ventana en X, pero está bien.

Sabiendo eso, le pasamos a las funciones auxiliares la imagen del muro a poner y la cantidad de muros que calculamos.

- **draw_muros_superiores_aux y draw_muros_inferiores_aux**


```

#Se creó una función recursiva que pone una imagen de forma recursiva hacia la derecha hasta que ya no caben, esto se hace haciendo la imagen
#para que sea el límite de la función recursiva.
#Función auxiliar #1
def draw_muros_superiores_aux(muro_imagen, cantidad_muros, posicion_x=0):#La posición_x, ponemos 0 para que empiece desde la izquierda del todo
    if cantidad_muros <= 0:
        return # Caso base: No quedan muros por dibujar ya que se llegó al final calculado

    # Dibujar un muro en el borde superior
    PANTALLA.blit(muro_imagen, (posicion_x, 0))

    # Calcular la posición x del siguiente muro
    nueva_posicion_x = posicion_x + muro_imagen.get_width() #Se le va sumando su ancho para que no quede una encima de la otra o separadas.

    # Llamar recursivamente a la función para dibujar el siguiente muro
    draw_muros_superiores_aux(muro_imagen, cantidad_muros - 1, nueva_posicion_x)

#Se tuvo que crear una función a parte solo para los muros inferiores ya que haciendolo todo solo con una principal y una auxiliar resultó en
#Función auxiliar #2
def draw_muros_inferiores_aux(muro_imagen, cantidad_muros, posicion_x=0):
    if cantidad_muros <= 0:
        return # Caso base: No quedan muros por dibujar

    # Dibujar un muro en el borde inferior
    PANTALLA.blit(muro_imagen, (posicion_x, 500))

    # Calcular la posición x del siguiente muro
    nueva_posicion_x = posicion_x + muro_imagen.get_width() # Avanzar una anchura de muro

    # Llamar recursivamente a la función para dibujar el siguiente muro
    draw_muros_inferiores_aux(muro_imagen, cantidad_muros - 1, nueva_posicion_x)

```

En resumidas cuentas, lo que hacen es recibir los parámetros anteriores más uno nuevo que se le otorga ahí mismo que es la posición en X, que es 0. Lo que se realiza es que se recursivamente se hace un calculo de sumatorias esto con el ancho de cada imagen que anteriormente dijimos que era 93, y las va apilando, recursivamente se encargan de sumar los largos de los bloques de manera que se vayan dibujando uno pegado al otro. El proceso es prácticamente lo mismo para los muros inferiores pero la diferencia es donde se dibujan, que es en la parte baja. El proceso termina cuando la cantidad de muros es menor o igual a 0, esto se logra haciendo que con cada llamada recursiva se reste 1 a la cantidad de muros correspondientes.

Específicamente para los muros indestructibles y destructibles:

```
#Lista de posiciones de los muros DESTRUCTIBLES(Si se puede, destruir pero no pasar por encima):
muros_destructibles_posiciones = [
    #Antes de línea #1:
    (0,50),
    (250,50),
    #línea #1
    (450,50),
    (0, 100),
    (0, 150),

    (150,150),
    (200,250),
    (250,250),
    (250,150),
    #Línea 3
    (500,300),
    #Debajo de Línea 3
    (400,350),
    (250,350),
    (200,350),
    (400,100),
    (400,150),
]

#Lista de posiciones de los muros indestructibles(No se puede destruir ni pasar por encima)
> muros_indestructibles_posiciones = [ ...
```

Primero se tienen la lista de muros destructibles e indestructibles que no son más que una serie de coordenadas dentro de una lista de Python.

```
#Sector Función recursiva encargada de darle imagen a los bloques internos(donde el jugador chocará jugando)colocados:
def draw_bloques_internos(posiciones):
    if not posiciones:
        return # Caso base: la lista de posiciones está vacía
    # Dibujar el bloque en la primera posición de la lista
    x, y = posiciones[0]
    PANTALLA.blit(muro_INTERNOS_scale,(x,y))
    # Llamar recursivamente a la función con la lista restante de posiciones de bloques
    draw_bloques_internos(posiciones[1:])

#Sector Función recursiva encargada de darle imagen a los bloques destructibles:
def draw_bloques_destructibles(posiciones):
    if not posiciones:
        return #Caso base: La lista de bloques por pintar está vacía
    #Se dibuja el bloque en la primera posición de la lista
    x,y = posiciones[0]
    PANTALLA.blit(muro_destructible_scale,(x,y))
    #Se llama recursivamente a la función con lo que queda por dibujar en la lista, eliminando el de atrás.
    draw_bloques_destructibles(posiciones[1:])

#-----Creación de los bordes exteriores(end)-----
```

Aquí se tienen dos funciones, las cuales se encargan de simplemente se encargan de dibujar los muros según las coordenadas en la lista según avanza de índice, hasta que ya no quedan posiciones que dibujar, de igual forma con los destructibles. El resto de la lógica de las colisiones con el jugador se explicó en la sección de movimiento del personaje, y la destrucción de los bloques destructibles se explica más adelante.

Explosivos y obstáculos:

Para esta sección se tiene funciones que interactúan con todo el código:

```
ejecuta = True #---> Básicamente lo que mantiene encendida esta máquina,
como la primera combustión de un carro, que mantiene encendido su motor hasta que se apaga.
def manejar_evento():
    global
    ejecuta,bombas_disponibles,bombas_disponiblesNivel2,bombas_disponiblesNivel3,llave_recogidaNivel1

    # Manejar un solo evento
    event = pygame.event.poll()
    if event.type == pygame.QUIT:
        ejecuta = False # Cuando se cierra la ventana desde la X
    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_SPACE:
            # Colocar una bomba en la posición actual del jugador y guardar las coords para que
            luego la bomba tenga donde plantarse y
            # luego comparar esas coords con los bloques.
            if nivel == 1: #Estos condicionales están conectados indirectamente con el contador de
            bombas, ya que el mismo muestra
                #cuentas bombas quedan disponibles en pantalla segun el nivel.
                #Variable #1 de bombas
                if bombas_disponibles > 0:
                    nueva_bomba = {"x": px, "y": py, "Indice de la imagen(posición en la lista)": 0,
                    "Tiempo que dura la bomba en juego": pygame.time.get_ticks()}
                    bombs.append(nueva_bomba)
                    bombas_disponibles = bombas_disponibles -1
                else:
                    mensaje = fuente.render("Ya no hay bombas disponibles", True, negro, blanco)
                    PANTALLA.blit(mensaje, (W // 2 - mensaje.get_width() // 2, H // 2 -
                    mensaje.get_height() // 2))
                    pygame.display.update() # Actualizar la pantalla para mostrar el mensaje
                    pygame.time.delay(1200) # Esperar 1.5 segundos antes de borrar el mensaje
                    PANTALLA.fill(verde) # Volver a llenar la pantalla para borrar el mensaje
```

Primero dentro de manejar_evento() cada vez que se presiona la tecla espacio, se extraen las coordenadas actuales del jugador("X" y "Y") y se le otorgan a una nueva variable llamada nueva_bomb, además se le da un índice en 0 para luego hacer las animación y además se obtiene el tiempo con pygame.time.get_ticks().

Luego esa variable que almacena las bombas se agrega a bombs que es una lista vacía para luego utilizarla. Nota* En manejar_evento(), hay condicionales limitando lo anterior dicho pero es por nivel, lo que permite poder hacer cálculos como restar la cantidad de bombas actuales cada vez que se le da espacio, es decir si estamos en nivel 1, entonces las bombas disponibles son 25, por lo tanto cada vez que se le da espacio resta 1, cuando llega a 0 aparece un mensaje informando que ya no hay bombas disponibles.

- **dibujar_contador_bombas():**

```
#CANTIDAD máxima de bombas Nivel 1:
bombas_disponibles = 25
bombas_disponiblesNivel2 = 20
bombas_disponiblesNivel3 = 24

# Función para dibujar el contador de bombas disponibles en la pantalla
def dibujar_contador_bombas():
    if nivel == 1:
        font = pygame.font.Font(None, 30)
        texto = font.render("Bombas disponibles: " + str(bombas_disponibles), True, negro)
        PANTALLA.blit(texto, (350, 580))

    if nivel == 2:
        font = pygame.font.Font(None, 30)
        texto = font.render("Bombas disponibles: " + str(bombas_disponiblesNivel2), True, negro)
        PANTALLA.blit(texto, (350, 580))

    if nivel == 3:
        font = pygame.font.Font(None, 30)
        texto = font.render("Bombas disponibles: " + str(bombas_disponiblesNivel3), True, negro)
        PANTALLA.blit(texto, (350, 580))
```

Esta función se encarga de dibujar el contador de bombas en pantalla según sea el nivel actual, esto se hizo así ya que tienen que haber una cierta cantidad de bombas por nivel, entonces, esta función toma el nivel actual, y la cantidad de bombas disponibles por nivel dibuja en pantalla lo correspondiente.

- **bomb_exploded(bomb):**

```
#Tiempo de vida de la bomba, antes, durante y después de explotar.(Tiempo que tiene para estar en el campo de juego antes de explotar)
def bomb_exploded(bomb):
    current_time = pygame.time.get_ticks()
    return current_time - bomb["Tiempo que dura la bomba en juego"] >= 3000
```

Realmente el funcionamiento de esta no es complejo, simplemente establece un tiempo máximo para que las bombas existan antes de que se inicie la secuencia de animación de explosión.

- **Explosión_bomba():**

Esta es la función principal de detectar_colision-bomba_bloque ya que le pasa la lista de muros destructibles como argumento, cada que la función bomb_exploded se activa o se llama, actualiza la lista de muros_destructibles_posiciones

En resumidas cuantas obtiene la bomba de la lista de bombs, a la bomba con el índice actual se le almacena en la variable bomb, luego se le dice, extraiga las coordenadas en X y Y, con estas coordenadas se comparará si la bomba actual está en el rango de explosión con el personaje, bloque, o enemigo.

- **detectar_colision_bomba_bloque (bomba_x, bomba_y, bloques, rango_colision=40):**

```
#Función auxiliar/recursiva de explosion bomba (Rango de explosión y demás)
def detectar_colision_bomba_bloque(bomba_x, bomba_y, bloques, rango_colision=40):
    if not bloques:
        return bloques # Caso base: si no hay bloques, devuelve la lista original

    # Extraer las coordenadas del primer bloque de la lista
    bloque_x, bloque_y = bloques[0]

    # Calcular el centro del bloque
    centro_bloque_x = bloque_x + 20 # Ajustar según el tamaño de los bloques
    centro_bloque_y = bloque_y + 20 # Ajustar según el tamaño de los bloques

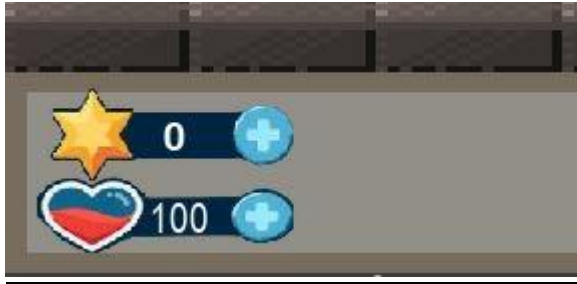
    # Verificar si la bomba está dentro del rango de colisión del bloque
    #En esta zona hubo un error que vale la pena anotar; Al poner valor absoluto a las coordenadas de las bombas y los bloques, al dar valores
    if abs(bomba_x - centro_bloque_x) <= rango_colision and abs(bomba_y - centro_bloque_y) <= rango_colision:
        # Si la bomba está dentro del rango de colisión, eliminar el bloque de la lista
        return bloques[1:]
    else:
        # Si la bomba no está dentro del rango de colisión, mantener el bloque en la lista
        return [bloques[0]] + detectar_colision_bomba_bloque(bomba_x, bomba_y, bloques[1:], rango_colision)
```

Esta es la función auxiliar del funcionamiento de las bombas con respecto a los bloques, como se explicó anteriormente en la sección de muros destructibles e indestructibles, estos están en una lista.

Esta función en específico recibe las coordenadas X y Y que anteriormente se extrajeron de la lista de bombas en la función principal y además recibe la lista de muros destructibles también un nuevo parámetro que es el rango de colisión de 40 pixeles, luego se hace algo similar a la función pasada, esta función se encarga de sacar/extraer las coordenadas de cada bloque y separarlas en X y Y, luego lo que se hace es utilizar dichas coordenadas y suamrles 20 que es la mitad faltante para llegar al centro del bloque tanto en X como en Y, luego se reutiliza la forma de calcular la colisión que hemos venido viendo, solo que si la bomba está lo suficientemente cerca del cetro del bloque, entonces elimínela

de la lista haciendo bloques[1:], sino está en el rango, entonces simplemente sume el actual + la llamada recursiva para analizar el siguiente muro.

Vida del jugador:



```
#Vida del personaje Se define acá para poder pasarla a la función explosion_bomba  
vida_personaje = 100
```

La vida del jugador no es más que una variable global que se mantiene en 100, amenos que se detecte una colisión con un enemigo o una bomba explote en el rango establecido y el jugador se encuentre ahí.

```
# Verificar si la bomba ha explotado con los ticks del juego, a lo Minecraft.  
if bomb_exploded(bomb):  
    if abs(px - bomb_x) <= 40 and abs(py - bomb_y) <= 40:  
        # Si el personaje está en el rango de la explosión, reducir su vida.  
        vida_personaje -= 5  
        print("¡La bomba ha explotado y has perdido 5 puntos de vida!")
```

Si el personaje está en el rango de explosión tanto en X como en Y, entonces resta 5 cada vez que se da la explosión

```
# Verificar si la vida ha llegado a 0 o menos
if vida_personaje <= 0:
    # Detener el bucle principal para salir del juego
    ejecuta = False

    # Mostrar mensaje de "Game Over"
    font = pygame.font.SysFont("Arial", 48)
    game_over_text = font.render("Game Over! You died", True, negro, blanco)
    text_rect = game_over_text.get_rect(center=(W // 2, H // 2))
    PANTALLA.blit(game_over_text, text_rect)
    pygame.display.update()

    # Pequeña pausa antes de salir del juego
    pygame.time.delay(2000) # Pausa de 2 segundos (2000 milisegundos)

    # Llamamos a la función que pasa los puntos al txt apenas finaliza el juego, para que así se actualice el bloc de notas
    agregar_puntos_a_txt(puntos)
    # Salir del bucle principal
    ejecuta = False
```

Si la vida del personaje llegara a 0, entonces el bucle principal para, se enseña en pantalla un mensaje, se ralentiza el tiempo para poder leerlo, se pasan los puntos almacenados a la función encargada de almacenarlos en el txt y finalmente acaba el ciclo.

```
# En este sector dibujamos la vida en pantalla y además una decoración (marco)
vidaDashboard = pygame.image.load("Imagenes/Health/Health Image.png")
vidaImagen_scale = pygame.transform.scale(vidaDashboard, (135, 45))
PANTALLA.blit(vidaImagen_scale, (13, 600))
fuente = pygame.font.SysFont("Arial", 21)
texto_vida = fuente.render(f"{vida_personaje}", True, blanco)
PANTALLA.blit(texto_vida, (73, 610))
```

De esta manera dibujamos en pantalla la vida actual y una imagen decorativa para una mejor visualización.

Distribución de Puntos en el Mapa:

La distribución de puntos fue lo último en ser desarrollado, de tal manera que alberga la cúspide empeño y conocimiento de este proyecto, en la bitácora se puede observar lo que se tardó en llevar a cabo esta parte, se reutilizó varias funcionalidades anteriormente usadas, por ejemplo, dibujar imágenes con las coordenadas de una lista para poder luego compararlas con la posición del jugador, anteriormente esto se realizó en los bloques destructibles e indestructibles, se procede a explicar el funcionamiento:

Primero se tiene una variable de puntos global, la cual está por defecto en cero (Se empieza en 0 porque no se han recolectado puntos aún):


```
#Variable puntos(  
puntos = 0
```

Se tienen varias listas de posiciones con los puntos por nivel, esto se hizo así para que cada vez que se cruce la puerta se logre mostrar una distribución diferente de puntos para ser más entretenido el juego:

```
# Lista de posiciones de los puntos y su respectivo nivel  
listaPosicionesPuntos = [  
    # Antes de Línea 1  
    (500, 50),  
    (550, 50),  
    #Línea 1  
    (100,100),  
    #Entre 1 y 2  
    (200,150),  
    #Línea 2  
    (400,200),  
    #Línea entre 2 y 3  
    (0,250),  
    (150,250),  
    (350,250),  
    (500,250),  
    #Línea 4  
    (100,400),  
    #Debajo línea 4  
    (150,450),  
  
]  
  
listaPosicionesPuntosNivel2 = [...]  
  
listaPosicionesPuntosNivel3 = [...]
```

Luego poseemos dos funciones, una encargada de dibujar los puntos por pantalla y la otra para detectar la colisión y eliminar ese punto del mapa cuando es recogido.

```
# Creamos una función que dibuja en pantalla la imágenes de los puntos
def dibujarPuntosPantalla(PosicionesPuntosenLista):
    if not PosicionesPuntosenLista:
        return #Si no hay puntos en la lista, retorne nada.

    x, y = PosicionesPuntosenLista[0] #Extraemos las coordenadas del punto a analizar, (x,y) en la lista.
    PANTALLA.blit(puntosImagen_scale, (x, y)) #Dibujamos
    dibujarPuntosPantalla(PosicionesPuntosenLista[1:])#Llamada recursiva analizando todos los componentes.
```

En esta se reutilizó el conocimiento adquirido en las otras partes de desarrollo del juego, adquiere las coordenadas de la lista de puntos, y las separa en X y Y, luego las va dibujando hasta que ya no queden puntos en la lista.

```
def detectar_colision_y_eliminar_puntos(px, py, listapuntos, puntos, rango_colision=40):
    if not listapuntos:
        return [], puntos # Si no quedan puntos en la lista, retorna la lista vacía y los puntos acumulados

    puntos_x, puntos_y = listapuntos[0] #Extraemos las coords del punto a analizar.
    centro_puntos_x = puntos_x + 20 #Ajustamos cual es el centro de, por decirlo así, el bloque del punto en X
    centro_puntos_y = puntos_y + 20# Lo mismo pero en Y

    if abs(px - centro_puntos_x) <= rango_colision and abs(py - centro_puntos_y) <= rango_colision:
        # Si el jugador colisiona con el punto, se suma un punto al contador y se elimina el punto de la lista
        print("Se sumó un punto")
        puntos += 1
        return listapuntos[1:], puntos
    else:
        # Si no hay colisión, se mantiene el punto en la lista
        nuevos_puntos, puntos_actualizados = detectar_colision_y_eliminar_puntos(px, py, listapuntos[1:], puntos)
        return [listapuntos[0]] + nuevos_puntos, puntos_actualizados
```

La de detección se basa en las pasadas, adquirieron las coordenadas de los puntos, calculando su centro, luego comparando su posición, si se detectó la colisión, entonces sume un punto y luego elimine esa coordenada de la lista de puntos. Si no se detectó colisión entonces simplemente se mantienen los puntos en este caso el punto actual se conserva y se agrega de nuevo a la lista de puntos restantes nuevos puntos y se continua el proceso con los puntos restantes.

Se realizaron 2 pares de funciones más que realizan exactamente lo mismo pero con el resto de listas de puntos es decir con la del nivel2 y 3, esto ya que no se encontró la manera de que un solo par de funciones pudiera manejar 3 listas de puntos diferentes.

Tipos de Enemigos:

Se cuentan con dos tipos de enemigos, en este caso el duende verde y el duende morado, el duende verde se dibuja dos veces.



Se definen las variables de vida y estado de los enemigos:

```
#Vida del duende morado:
vida_DuendeMorado = 50
#Variable del duende verde
vida_DuendeVerde = 50
#Vida del duende verde Vertical
vida_DuendeVerdeVertical = 50
# Variable para controlar si el duende morado está vivo o muerto
duende_morado_vivo = True
duende_verde_vivo = True
duende_verdeVertical_vivo = True
#
```

```
#Si una bomba explota a la par del duende morado, le baja vida, cuando llega a 0, muere(se frena la función de
# dibujado, y desaparece)
if (duende_morado_x1 - bomb_x) <= 40 and abs(duende_morado_y1 - bomb_y) <= 40 and nivel == 3:
    # Si el personaje está en el rango del duende, reducir su vida
    vida_DuendeMorado -= 10
    print("Poner sonido de muerte del duende")
    if vida_DuendeMorado <= 0:
        duende_morado_vivo = False

if (duendeVerde_x1 - bomb_x) <= 40 and abs(duendeVerde_y1 - bomb_y) <= 40 and nivel == 2:
    # Si el personaje está en el rango del duende, reducir su vida
    vida_DuendeVerde -= 10
    print("Poner sonido de muerte del duende")
    if vida_DuendeVerde <= 0:
        vida_DuendeVerde = False

if (duendeVerdeVertical_x1 - bomb_x) <= 40 and abs(duendeVerdeVertical_y1 - bomb_y) <= 40 and nivel == 3:
    # Si el personaje está en el rango del duende, reducir su vida
    vida_DuendeVerdeVertical -= 10
    print("Poner sonido de muerte del duende")
    if vida_DuendeVerdeVertical <= 0:
        vida_DuendeVerdeVertical = False
```

Se ha de mencionar que los enemigos se ven afectados por las bombas, anteriormente se explicó que en la función explosión_bomba() como funciona esto, pero es bajo el mismo principio que el jugador principal.

Movimiento y animación de los enemigos

```
#----- Movimiento de los enemigos(Skins)"Inicio" -----#
# Variable del movimiento enemigo duende morado (Quieto)
quietoDuendeMorado = pygame.image.load("Imágenes/Enemigos/DuendeMorado/Quieto/Quieto.png")
# Camina hacia arriba
CaminaHaciaArribaDuendeMorado = [pygame.image.load("Imágenes/Enemigos/DuendeMorado/Up/1.png"),
                                   pygame.image.load("Imágenes/Enemigos/DuendeMorado/Up/2.png"),
                                   pygame.image.load("Imágenes/Enemigos/DuendeMorado/Up/3.png")]
# Camina hacia abajo
CaminaHaciaAbajoDuendeMorado = [pygame.image.load("Imágenes/Enemigos/DuendeMorado/Down/1.png"),
                                  pygame.image.load("Imágenes/Enemigos/DuendeMorado/Down/2.png"),
                                  pygame.image.load("Imágenes/Enemigos/DuendeMorado/Down/3.png")]

#-----para el duende 2-----
# Variable del movimiento enemigo duende morado (Quieto)
quietoDuendeVerde = pygame.image.load("Imágenes/Enemigos/DuendeVerde/Quieto/Quieto.png")
# Camina hacia arriba
CaminaHaciaDerechaDuendeVerde = [pygame.image.load("Imágenes/Enemigos/DuendeVerde/Right/1.png"),
                                   pygame.image.load("Imágenes/Enemigos/DuendeVerde/Right/2.png"),
                                   pygame.image.load("Imágenes/Enemigos/DuendeVerde/Right/3.png"),
                                   ]
# Camina hacia abajo
CaminaHaciaIzquierdaDuendeVerde = [ pygame.image.load("Imágenes/Enemigos/DuendeVerde/Left/1.png"),
                                      pygame.image.load("Imágenes/Enemigos/DuendeVerde/Left/2.png"),
                                      pygame.image.load("Imágenes/Enemigos/DuendeVerde/Left/3.png"),
                                      ]

CaminaHaciaArribaDuendeVerde = [
    pygame.image.load("Imágenes/Enemigos/DuendeVerde/Up/1.png"),
    pygame.image.load("Imágenes/Enemigos/DuendeVerde/Up/2.png"),
    pygame.image.load("Imágenes/Enemigos/DuendeVerde/Up/3.png"),
]

CaminaHaciaAbajoDuendeVerde = [
    pygame.image.load("Imágenes/Enemigos/DuendeVerde/Abajo/1.png"),
    pygame.image.load("Imágenes/Enemigos/DuendeVerde/Abajo/2.png"),
    pygame.image.load("Imágenes/Enemigos/DuendeVerde/Abajo/3.png"),
]
```

```

# Velocidad a la que se mueve el enemigo Verde
velocidadDuendeVerde = 2

# Velocidad a la que se mueve el enemigo Morado
velocidadDuendeMorado = 2

# Velocidad a la que se mueve el enemigo Verde Vertical
velocidadDuendeVerde_Vertical = 2

#-----para el duende 2-----

# Coordenadas iniciales del enemigo duende morado
duende_morado_x1 = 300
duende_morado_y1 = 280

duendeVerde_x1 = 100
duendeVerde_y1 = 450

# Índice de la imagen actual para la animación del duende
indice_animacion_duende_morado = 0

# Índice de la imagen actual para la animación del duende
indice_animacion_duende_Verde = 0

# Variable para controlar el movimiento inicial del duende morado
duende_morado_movimiento = "arriba" # Movimiento inicial del duende morado

duende_verde_movimiento = "derecha"

```

Lo anterior corresponde a las imágenes e índices de animación para que cuando el enemigo se mueva, este se anime de la misma forma que lo hace el jugador.

Se hace ilusión a las siguientes funciones encargadas de animar a los enemigos y además de detectar la colisión enemigo-jugador:

```

# Función para actualizar la posición del duende morado según el ciclo de movimiento predefinido
def actualizar_posicion_duende_morado():
    global duende_morado_x1, duende_morado_y1, vida_personaje, vida_DuendeMorado, duende_morado_movimiento

    # Verificar si estamos en el nivel 2 para agregar duendes morados
    if vida_DuendeMorado > 0:
        # Actualizar la posición del duende morado según su dirección de movimiento y velocidad
        if duende_morado_movimiento == "abajo":
            duende_morado_y1 += velocidadDuendeMorado
        elif duende_morado_movimiento == "arriba":
            duende_morado_y1 -= velocidadDuendeMorado
        # Cambiar la dirección del duende morado cuando alcanza ciertos límites
        if duende_morado_y1 >= 280:
            duende_morado_movimiento = "arriba"
        elif duende_morado_y1 <= 50:
            duende_morado_movimiento = "abajo"

        if abs(px - duende_morado_x1) <= 40 and abs(py - duende_morado_y1) <= 40:
            # Si el personaje está en el rango del duende, reducir su vida
            vida_personaje -= 2
            print("¡Duende!")
    else:
        vida_DuendeMorado = False

```

En esta función se determina hasta donde va a caminar tanto arriba como abajo, también se condiciona la colisión contra el jugador.

```

# Función para dibujar todos los duendes morados en sus posiciones actuales
def dibujar_duende_morado():
    global duende_morado_x1, duende_morado_y1, indice_animacion_duende_morado

    # Verificar si estamos en el nivel 2 para dibujar duendes morados
    # Obtener la imagen correspondiente al movimiento actual del duende morado
    imagen = None
    if duende_morado_movimiento == "arriba":
        imagen = CaminaHaciaArribaDuendeMorado[indice_animacion_duende_morado]
    elif duende_morado_movimiento == "abajo":
        imagen = CaminaHaciaAbajoDuendeMorado[indice_animacion_duende_morado]
    # Escalamos la imagen del duende
    imagen_redimensionada = pygame.transform.scale(imagen, (40, 40))
    # Dibujar al duende morado en su posición actual
    PANTALLA.blit(imagen_redimensionada, (duende_morado_x1, duende_morado_y1))
    # Incrementar el índice de la animación para la próxima imagen
    indice_animacion_duende_morado = (indice_animacion_duende_morado + 1) % len(CaminaHaciaArribaDuendeMorado)

```

En esta se permite dibujar al duende y sus respectivas imágenes para da movimiento cuando camina hacia arriba o abajo.

Ambas funciones acuden a las variables y listas anteriormente descritas.

Esto al igual como los puntos, se tienen funciones para dibujar diferentes duendes, no se encontró la manera de hacer que un solo par de funciones dibujaran varios duendes con diferente comportamiento.

Dificultades Encontradas

Empezando el desarrollo del proyecto se encuentra el reto de como adaptar un bucle for hacía la recursividad, es decir, entender cómo aplicar la recursividad para mantener un bucle, ya que es lo que se solicita como funcionalidad principal. Durante la adaptación hacía la recursividad se encontró la situación en la cual se superaba la cantidad máxima de llamadas recursivas admitidas (999), haciendo que la ventana del juego no se mantuviera estable no más de 20 segundos. Esto se resolvió aplicando un delay, o lentitud.

```
Traceback (most recent call last):
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra, Bomberman\Bomberman Code\Game\bomberman.PY", line 176, in bucle_principal
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra, Bomberman\Bomberman Code\Game\bomberman.PY", line 176, in bucle_principal
    bucle_principal()
[Previous line repeated 995 more times]
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra, Bomberman\Bomberman Code\Game\bomberman.PY", line 117, in bucle_principal
    manejar_evento()
RecursionError: maximum recursion depth exceeded
```

Durante la etapa del desarrollo del código principal del juego, estrictamente en las mecánicas de movimiento del personaje, al integrar las animaciones de movimiento en el bucle principal provocó una serie de problemas, el personaje no aparecía en pantalla, o aparecía únicamente cuando se mantenían pulsadas las teclas, también la limitación de movimiento de dicho personaje no concordaba con los márgenes de la pantalla he hizo que no se moviera correctamente.

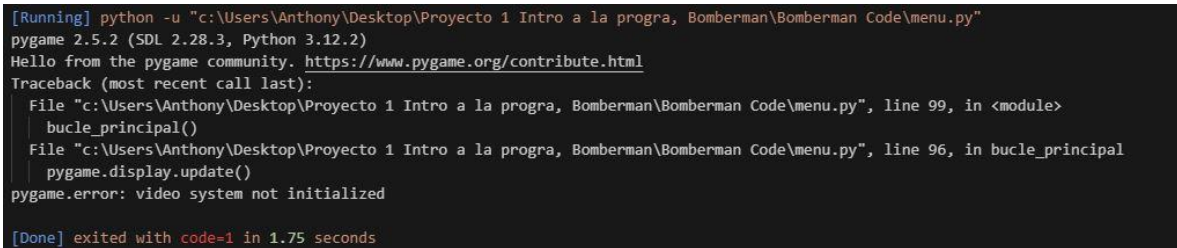
El código se encontraba desordenado, por lo cual una vez se repararon las dificultades anteriores, se procedió a ordenar y comentar las líneas de código para proseguir.

Durante el desarrollo de la pantalla menú, se crearon múltiples botones para diferentes fines, iniciar el juego, configuraciones y demás, entre esos el botón “salir”, este último presentó un problema, al presionarlo daba un error (INSERTAR IMAGEN) el cual para solucionarse se tuvo que recurrir a cambiar la función a la cual estaba llamando y además en crear una nueva función especializada en cerrar la ventana cuando dicho botón se presionaba.

Anteriormente en la parte de la lista de botones, el botón número 4 que simbolizaba cerrar el juego, cuando era presionado realizaba la acción “pygame.quit()” Lo que provocaba que se intentara cerrar el juego antes de que pygame hubiera completado su inicialización, para solucionarlo se le dio la posibilidad al botón 4 de llamar una nueva función comentada

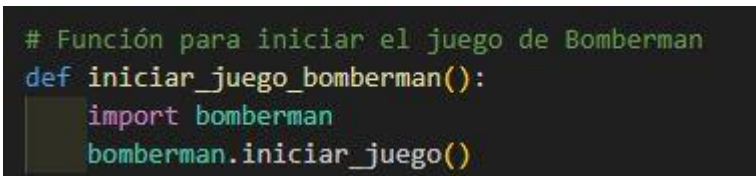
anteriormente, que se especializa en cerrar el juego cuando el mismo está iniciado y es llamada, definiendo el botón 4 de la siguiente manera: “(button_4, lambda: salir_del_menu())” y a la función encargada de cerrar el juego

```
def salir_del_menu ():  
    if pygame.get_init ():  
        pygame.quit ()  
        quit ()”.
```



```
[Running] python -u "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra, Bomberman\Bomberman Code\menu.py"  
pygame 2.5.2 (SDL 2.28.3, Python 3.12.2)  
Hello from the pygame community. https://www.pygame.org/contribute.html  
Traceback (most recent call last):  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra, Bomberman\Bomberman Code\menu.py", line 99, in <module>  
    bucle_principal()  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra, Bomberman\Bomberman Code\menu.py", line 96, in bucle_principal  
    pygame.display.update()  
pygame.error: video system not initialized  
[Done] exited with code=1 in 1.75 seconds
```

Se presentó una dificultad al intentar anclar el juego menú del principal, cuando se ejecutaba el menú en lugar de mostrar la ventana de botones para seleccionar una opción, se ejecutaba primero el juego, sin dar la posibilidad de elegir algún botón del inicio... La solución fue importar el juego dentro de la función especializada de iniciar el juego cuando dicha función era llamada, en lugar de importar el juego directamente por fuera, ahora el juego solo se ejecuta si el botón 1 es presionado, lo cual llama la función iniciar juego para posteriormente importar el código del juego.



```
# Función para iniciar el juego de Bomberman  
def iniciar_juego_bomberman():  
    import bomberman  
    bomberman.iniciar_juego()
```

Se encontró un error en el código que provocaba que el personaje principal no apareciera correctamente sobre el campo de juego, sino que apareciera y desapareciera constantemente

Durante el desarrollo se encontró una dificultad bastante notoria y fue la necesidad de crear dentro del terreno de juego espacios(Bloques) donde el jugador no pudiera caminar porque representaban paredes, utilizando bucles itinerantes como “for” este problema se resolvería sencillamente, pero en esta ocasión todo tuvo que ser resuelto con recursividad, se crearon

múltiples funciones tanto auxiliares como principales que se dedicaban a diferentes fines pero todas en conjunto lograron crear espacios donde el jugador no pudiera caminar, y además se corrigió un error en el no dejaba poner una imagen encima del bloque delimitador.

Se dedicó una gran cantidad de tiempo en intentar crear la mecánica en la cual las bombas eliminaran(explotaran) bombas, aun así, no fue posible hacer que la bomba eliminara el bloque que estuviera debajo de ella, este error sería arreglado más adelante.

El día 30/03/2024 se decidió no avanzar con el proyecto hasta recibir asesoría del profesor encargado, ya que la implementación de algunas mecánicas no estaba del todo clara y se prefirió acudir a consulta para revisión general del código y aclaración de dudas.

El día 01/04/2024 Se prosiguió con el proyecto, pero se encontró la siguiente limitación: Al crearse un menú donde la persona pudiera seleccionar una de 3 skins surgió un problema y fue que las imágenes de las skins fueron migradas al código del submenú, donde una vez la persona seleccionara la skin, dichas imágenes correspondientes se pasaran como argumento a una función que importaba el juego y le pasa las imágenes a utilizar para el movimiento del personaje, pero no se logró dicho cometido.

El día 02/04/2024 Durante este día se intentó corregir nuevamente el menú, pero sin tener éxito.

El día 03/04/2024 Se decidió detener el desarrollo del menú de personalización, por recomendación del profesor encargado, se decidió seguir con la programación de las siguientes partes pendientes y decir menú o descartado, o para más adelante.

Durante el desarrollo de la lógica para encontrar, recoger y usar una llave dispersa por el mapa, se necesitaba esconder debajo de un bloque destructible “Box”, pero la llave aparecía super puesta por delante del bloque, esta dificultad se corrigió dibujando antes que los bloques las funciones de la llave y puerta en el bucle del juego.

El día 05/04/2024 3h Se intentó implementar la mecánica de los niveles, pero no fue posible, la manera en la que está creado el juego no permite la creación de niveles de una manera sencilla.

06/04/2024 7h y 30 minutos Este día se dedicó a la investigación de como implementar diferentes niveles y la posible programación de esto (No fue posible).

07/04/2024 Se intentó desarrollar el sistema de niveles nuevamente pero no fue posible. Se modificó la manera en cómo se llaman las funciones al código principal, esto con el fin de poder entrar y salir de las ventanas cuantas veces quiera, anteriormente solo se podía entrar y salir una vez, el error fue que al momento de llamar a las funciones no se les estaba poniendo “()” al final, ejemplo de una función que tenía este problema:

```
def iniciar_juego_bomberman():  
    import Bomberman  
    Bomberman.bucle_principal()
```

09/04/2024 Se logró encontrar la solución al problema de los niveles, específicamente que la puerta al tener colisión con ella, no permitía el dibujo de los duendes según el nivel, sin embargo, realmente todo estaba bien, el gran problema fue que la puerta no tenía un limitador de colisiones, lo que quiere decir que cuando chocaba por primera vez(llevando la llave recogida), se aumentaba de nivel, pero a la vez seguía subiendo un nivel por cada segundo que estaba en colisión con la puerta, lo que resultaba que si se dibujaran los enemigos del nivel 2, pero pasara de largo rápidamente, solución, limitar la colisión de la puerta a 1, haciendo que se pudieran dibujar correctamente los enemigos del nivel correspondiente.

El día 10/04/2024 surgieron varios problemas a la hora de desarrollar el sistema de puntos ya que las funciones no llegaban a dibujar o contar los puntos en pantalla, uno de los problemas más grandes a resaltar es que había dos funciones que por separado se dedicaban a detectar las colisiones Jugador-Punto y la hora a eliminar dicho punto si hubo colisión, pero debido a la cantidad de errores se decidió fusionar ambas funciones en una única para poder tener un mejor manejo y organización.



El día 12/04/2024, se logró arreglar un error que no permitía mantener la ventana de menú de selección de skins y nombre abierta por más de 10 segundos, el error se debía a la forma se había construido la función del bucle principal, se olvidó implementar un bucle while que mantuviera las acciones, entonces todo se realizaba manejando los eventos en un poll, pero al final esto era ineficiente, se corrigió con éxito, el problema por consola era:

```
[Running] python -u "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra,  
Bomberman\Bomberman Code\Bomberman.py"  
pygame 2.5.2 (SDL 2.28.3, Python 3.12.2)  
Hello from the pygame community. https://www.pygame.org/contribute.html  
Traceback (most recent call last):  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra,  
Bomberman\Bomberman Code\Bomberman.py", line 130, in <module>  
    manejar_evento()  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra,  
Bomberman\Bomberman Code\Bomberman.py", line 126, in manejar_evento  
    manejar_evento()  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra,  
Bomberman\Bomberman Code\Bomberman.py", line 126, in manejar_evento  
    manejar_evento()  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra,  
Bomberman\Bomberman Code\Bomberman.py", line 126, in manejar_evento  
    manejar_evento()  
[Previous line repeated 990 more times]  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra,  
Bomberman\Bomberman Code\Bomberman.py", line 99, in manejar_evento  
    if dibujar_boton("Iniciar juego", 250, 400, 150, 50, verde, blanco):  
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
  File "c:\Users\Anthony\Desktop\Proyecto 1 Intro a la progra,  
Bomberman\Bomberman Code\Bomberman.py", line 63, in dibujar boton
```


Seguidamente se procedió a seleccionar, editar y recortar, adaptar, las imágenes o “Sprites” del personaje controlable del juego y los enemigos, siendo así las imágenes que darán movimiento a dichas entidades en los ejes verticales y horizontales de la pantalla de juego.

En otra instancia se inició con el código principal para empezar a desarrollar el juego, como descargar, importar e inicializar pygame. Posteriormente se definieron las proporciones de la pantalla, un fondo a base de ejemplo, y las mecánicas básicas del movimiento del personaje jugable, caminar hacia arriba, abajo, derecha e izquierda.

La implementación de la recursividad para crear el bucle principal del juego, el cual mantiene la ventana del juego de forma permanente mientras dicho programa se mantenga activo, a la vez, la recursividad dentro del bucle otorgó la funcionalidad de poner cerrar la ventana del juego cuando se desea cesar operaciones.

Posteriormente se resolvieron algunos inconvenientes con el desarrollo, planteamiento y compatibilidad de la recursividad con el resto del código, esto mencionado en la sección de Dificultades encontradas, una vez se realizó dicho paso, se organizó y comentó cada parte del código para mantener el orden y proseguir con un desarrollo más limpio.

28/03/24

Se logró anclar el código del juego principal con el menú, pudiendo así iniciar el menú para posteriormente hacer clic en “Iniciar Juego” y ejecutar el código del juego.

Se tuvo que cambiar la manera en la que el personaje se movía ya que los bloques sólidos (Paredes) donde el jugador no puede avanzar cambió algunas formas en las que las condiciones del movimiento se veían afectadas.

Se creó una función recursiva con dos funciones auxiliares, estas tres en conjunto se destinaron para colocar bloques en la parte inferior y superior de la pantalla de forma que fueran consecutivas y formaran una pared con la temática del fondo.

Se reestructuró el código para corregir errores como que se dibujara antes el fondo que los bloques o el personaje, todo esto para que la eficiencia y funcionamiento del juego

fuera optimo, además se recurrió a comentar las nuevas zonas del código para orientarse mejor en las diferentes secciones.

29/03/2024

Se implementó la mecánica de bombas, específicamente la animación y la funcionalidad de poner varias de ellas cada cierto tiempo. También se logró que pudieran eliminar(explotar) ciertos bloques, pero no fue posible hacer que se elimina el que estuviera justo debajo de la bomba, solo el de arriba, izquierda y derecha.

30/03/2024, Este día no se prosiguió con la programación del código debido a dudas acerca de cómo implementar ciertas mecánicas, este día se dedicó a investigación y asesoría para poder seguir con una construcción limpia y ordenada del código hacia las próximas semanas.

31/03/2024

Este día se logró arreglar un pequeño error “bug”, que no permitía que los bloques tanto destructibles como indestructibles tuvieran sus imágenes correspondientes, solamente aparecían con la forma de un cuadrado con colores simples, pero el error fue solucionado y ahora muestran las skins correspondientes a los bloques; Destructibles (cajas “box”) e Indestructibles (bloques “Blocks”).

El día 01/04/2024 Se logró crear un submenú donde la persona podía ingresar su nombre y seleccionar una de tres skins disponibles para jugar, el nombre se almacena en un archivo texto, sin embargo, no fue posible conectar ese menú e imágenes correspondientes a la selección de la persona con el código principal del juego... este problema se arregló más adelante.

El día 02/04/2024 No se avanzó más debido a que no se pudo unir el menú con el submenú donde se seleccionaba la skin y se escribía el nombre, se descubrió que el problema era mucho mayor de lo que se esperaba, por lo tanto, se decidió dejar este menú

de selección de skin y nombre de último para poder avanzar con los otros puntos restantes, solamente se implementó la opción de limitar la cantidad de bombas.

El día 03/04/2024 Se avanzó bastante con el desarrollo, se logró establecer una variable “vida” para el personaje jugable, dicha estadística ahora se ve afectada por las explosiones de las bombas del mismo jugador y también la colisión con los enemigos, esto último fue otro desarrollo de este día, se logró implementar un enemigo “Duende Morado” el cual tiene una caminata predefinida y si llega a colisionar con el jugador, resta puntos a la vida. También se logró incluir una puerta la cual se llega a abrir con una llave dispersa alrededor del mapa, dicha llave se encuentra “detrás” de los bloques destructibles “box”.

El día 04/04/2024 Se lograron implementar las siguientes mecánicas: Ahora las bombas les hacen daño a los enemigos y estos desaparecen al llegar a vida 0, el tiempo, bombas disponibles y tiempo de juego, ahora aparecen en pantalla además cuando se recoge la llave escondida, esta se dibuja en el “inventario” del jugador.

El día 05/04/2024 3h Se intentó implementar la mecánica de los niveles, pero no fue posible, la manera en la que está creado el juego no permite la creación de niveles de una manera sencilla.

06/04/2024 7h y 30 minutos Este día se dedicó a la investigación de como implementar diferentes niveles y la posible programación de esto (No fue posible).

07/04/2024 Se desarrolló la pantalla de información del creador, se implementaron diferentes botones para poder volver a la pantalla menú principal, se intentó implementar diferentes niveles, esto último sin éxito.

El día 08/04/2024 Se logró hacer una ventana de mejores puntajes, que muestra por pantalla los nombres ingresados en la ventana pos-juego, y un puntaje de ejemplo ya que la lógica de los puntos no está implementada aún, además se mejoró la estética de algunas ventanas.

09/04/2024 Se logró encontrar la solución al problema de los niveles, específicamente que la puerta al tener colisión con ella, no permitía el dibujo de los duendes según el nivel, sin embargo, realmente todo estaba bien, el gran problema fue que la puerta no tenía un limitador de colisiones, lo que quiere decir que cuando chocaba por primera vez(llevando la llave recogida), se aumentaba de nivel, pero a la vez seguía subiendo un nivel por cada segundo que estaba en colisión con la puerta, lo que resultaba que si se dibujaran los enemigos del nivel 2, pero pasara de largo rápidamente, solución: limitar la colisión de la puerta a 1, haciendo que se pudieran dibujar correctamente los enemigos del nivel correspondiente. Correspondiente al desarrollo de los niveles se logró establecer una puerta y llave por nivel, siendo así que ahora hay una puerta y llave única, dando la sensación de progresión.

Este día también se logró hacer tres variables de bombas para cada nivel y que las funciones de “mostrar bombas disponibles” y “manejar evento” pudieran establecer un máximo y mínimo de bombas por nivel, permitiendo tener las mismas limitadas.

10/04/2024 Se logró hacer la pantalla final de ganar cuando se cruza la última puerta. Se organizó el código una vez más.

Se desarrolló que cuando se cruce la puerta de un nivel, emerja un mensaje indicando el nivel correspondiente.

Se logró desarrollar un sistema de puntos que cuando se detecte una colisión con la imagen de un punto, se sume un valor de 1 a una variable, se reutilizaron mecanismos de funciones usadas anteriormente como la creación de bombas, bloques destructibles y su eventual eliminación , la creación y de llaves, todo esto se reutilizó para crear unas funciones específicas para la creación de puntos en el mapa bajo el mismo funcionamiento de las anteriores descritas, queda pendiente elaborar un contador de puntos y que este se muestre en pantalla, además que el puntaje se almacene en un “txt” que posteriormente sea leído por la ventana de mejores puntajes.

10/04/2024 Este día se realizaron múltiples actividades, en la sección de “Estadísticas de tiempos” se puede consultar que se realizó este día.

11/04/2024 Al igual que el día anterior se realizaron varias actividades, se puede consultar en el apartado de “Estadísticas de tiempo”

12/04/2024 Se logró corregir un error que no permitía a la ventana de selección de skin y nombre mantenerse activa por más de 10 segundos.

13/04/2024 Este día se realizaron múltiples correcciones y modificaciones, se puede consultar en el apartado de “Estadísticas de tiempo”

14/04/2024 Como último día de desarrollo, se reorganizó el código, se comentaron partes de este donde se podía mejorar la explicación.

Se le dieron últimos detalles al juego.

Se terminó la documentación externa.

Estadística de Tiempos

Fecha	Actividad
26/03/2024 2 horas	Organizar instrucciones e información inicial para el desarrollo del proyecto.
26/03/2024 2 horas	Se inicia con el trabajo escrito y la documentación correspondiente
27/03/2024 16 horas	Se inicia el desarrollo del juego instalando pygame e importándolo, posteriormente se desarrollan los códigos base para empezar. Se definieron las proporciones de la pantalla, un fondo a base de ejemplo, y las mecánicas básicas del movimiento

	del personaje jugable, caminar hacia arriba, abajo, derecha e izquierda.
28/03/24 18 horas	<p>Se desarrolla un menú básico con recursividad, para posteriormente anclar el código del juego y poder ser inicializado desde el botón “Iniciar Juego”.</p> <p>Se desarrolla la mecánica de colisiones para delimitar espacios dentro del área de juego donde el personaje no pueda caminar (Paredes).</p> <p>Se decoró el fondo del juego y se organizó el código comentando cada parte y reestructurando sectores.</p>
29/03/2024 8 horas	Se implementó la mecánica de bombas, específicamente la animación y la funcionalidad de poner varias de ellas cada cierto tiempo. También se logró que pudieran eliminar(explotar) ciertos bloques, pero no fue posible hacer que se elimina el que estuviera justo debajo de la bomba, solo el de arriba, izquierda y derecha
30/03/2024 2hs	Asesoramiento e investigación de como implementar algunas mecánicas a futuro.
31/03/2024 2hs y media	Se logró arreglar un error “Bug” relacionado a que los bloques tanto destructibles como indestructibles no

	<p>mostrarán sus imágenes correspondientes.</p> <p>Se comentó y organizó el código otra vez para mayor legibilidad y orientación debido a que el código ya cuenta con más de 400 líneas de código.</p>
01/04/2024 3 horas y media	<p>Creación de un submenú donde la persona podría seleccionar una skin e ingresar su nombre, dicho menú fue descartado por el momento ya que no se logró conectar correctamente con el resto de scripts del juego.</p>
02/04/2024 1h y media	<p>Este día no se siguió trabajando en el desarrollo del juego a razón de esperar una reunión con el profesor encargado para hablar sobre el menú que no se podía anclar al resto de códigos, dicha reunión se daría al día siguiente, por lo tanto, este día se dedicó a intentar reparar los errores</p>
03/04/2024 7h y media	<p>Este día se realizó la reunión con el profesor encargado y se llegó a la conclusión de descartar la reparación, uso y adaptación del menú “seleccionarSkinNombre”, ya que el intento de reparar dichos errores o “bugs” iba a conllevar invertir demasiado tiempo, por lo tanto se descartó por el momento, de este modo: Se continuó con el desarrollo del juego,</p>

	<p>se logró desarrollar “vida” para el personaje jugable y un enemigo el cual si tiene contacto con el jugador resta vida, además cuando la vida llega a “0” se muestra un mensaje por pantalla de “Gamer Over”. Por último, se logró desarrollar la lógica correspondiente para crear una llave situada en un lugar oculto del mapa, cuya función es la de abrir una puerta que conduce a otro nivel.</p>
04/04/2024 3hs y media	<p>Se lograron implementar las siguientes mecánicas: Ahora las bombas les hacen daño a los enemigos y estos desaparecen al llegar a vida 0, el tiempo, bombas disponibles y tiempo de juego, ahora aparecen en pantalla además cuando se recoge la llave escondida, esta se dibuja en el “inventario” del jugador.</p>
05/04/2024 3h	<p>Se intentó implementar la mecánica de los niveles, pero no fue posible.</p>
<u>06/04/2024 7h y 30 minutos</u>	<p>Investigación de como implementar diferentes niveles y la posible programación (<u>No</u> fue posible), se realizó una ventana de configuración donde el jugador puede deshabilitar o habilitar música antes de jugar.</p>

<p><u>07/04/2024 8h</u></p>	<p>Se intentó implementar diferentes niveles, pero otra vez sin éxito, se desarrolló la pantalla de información del creador y además se corrigió un error que no permitía ingresar más de una vez a la misma pantalla.</p> <p>Se logró hacer que estando en el menú cuando se selecciona la opción “Iniciar juego” ahora se pide ingresar un nombre y seleccionar una skin, dicho nombre se almacena en un archivo txt.</p>
<p><u>08/04/2024 3h 30minutos</u></p>	<p>Se creó una ventana anclada al menú, dicha ventana es la de mejores promedios, a pesar de no tener la lógica de los puntos a la hora de jugar, se desarrolló dicha ventana. Se mejoró la estética visual de la ventana pos-juego donde se seleccionar el nombre y la skin, se colocó una imagen encima de cada botón con su correspondiente skin.</p>
<p>09/04/2024 6h</p>	<p>Se logró solucionar el problema de la puerta, que no dibujaba correctamente los enemigos del nivel correspondiente, se procede a desarrollar los diferentes niveles. Correspondiente al desarrollo de los niveles se logró establecer una puerta y llave por nivel, siendo así que ahora hay una puerta y llave única, dando la sensación de progresión. .se</p>

	<p>logró hacer tres variables de bombas para cada nivel y que las funciones de “mostrar bombas disponibles” y “manejar evento” pudieran establecer un máximo y mínimo de bombas por nivel, permitiendo tener las mismas limitadas.</p> <p>También se volvió a reestructurar el código y a comentar las nuevas zonas para una mejor navegación y orientación.</p>
10/04/2024 4h	<p>Se logró hacer la pantalla final de ganar cuando se cruza la última puerta.</p> <p>Se desarrolló que cuando se cruce la puerta de un nivel, emerja un mensaje indicando el nivel correspondiente.</p> <p>Se logró desarrollar un sistema de puntos que cuando se detecte una colisión con la imagen de un punto, se sume un valor de 1 a una variable, se reutilizaron mecanismos de funciones usadas anteriormente como la creación de bombas, bloques destructibles y su eventual eliminación , la creación y de llaves, todo esto se reutilizó para crear unas funciones específicas para la creación de puntos en el mapa bajo el mismo funcionamiento de las anteriores descritas, queda pendiente elaborar un</p>

	<p>contador de puntos y que este se muestre en pantalla, además que el puntaje se almacene en un “txt” que posteriormente sea leído por la ventana de mejores puntajes.</p> <p>Por último, se organizó el código una vez más.</p>
11/04/2024 3h	<p>Se implementó un contador que muestra en pantalla los puntos recogidos hasta el momento, además se desarrolló que los puntos, una vez finalizada la partida, ya sea “Game Over” o “Ganaste”, se llame a una nueva fusión encargada de introducir los puntos almacenados en la variable puntos en un txt, de forma que luego ese txt lo pueda leer la ventana lectora de puntos.</p> <p>Se aplicó el método usado para dibujar enemigo, para dibujar puntos almacenados en distintas listas, siendo así que según el nivel unos condicionales en el bucle del juego permiten o no dibujar las diferentes listas, esto con el propósito de hacer que la persona jugadora logre sentir progresión a medida que el juego avanza, dibujando puntos en diferentes posiciones.</p>

12/04/2024 1h	Se logró corregir un error que no permitía a la ventana de selección de skin y nombre mantenerse activa por más de 10 segundos.
13/04/2024 <u>10h</u>	<p>Se logró hacer un botón para volver al menú principal en la ventana de mejores puntajes.</p> <p>Se completó la lista de puntos según nivel, es decir ya todos los puntos(recolectables) están puestos de tal forma que se dibujan en pantalla.</p> <p>También se logró poner una imagen de barra de vida y la de puntos, de tal manera que sea más agradable.</p> <p>Se cambió el aspecto estético de las ventanas en general, como la del menú principal, configuración y mejores puntajes, con el fin de que sea más atractivo (Anteriormente solo tenían un fondo blanco y botones negros).</p> <p>En este día se organizó el código con el fin de arreglar algunos aspectos defectuosos como comentarios que nunca se eliminaron y ya no tienen función y también la corrección de errores.</p> <p>Además, para la decoración de las diferentes ventanas se invirtió tiempo considerable en editar las diferentes</p>

	imágenes para darle un aspecto original al diseño.
14/04/2024 5h y 30min	<p>Como último día de desarrollo, se reorganizó el código, se comentaron partes de este donde se podía mejorar la explicación.</p> <p>Se le dieron últimos detalles al juego.</p> <p>Se terminó la documentación externa.</p>
Total, tiempo: 118h	Se concluyó el desarrollo del video juego.

Conclusión

Durante el desarrollo de este proyecto se afrontaron, discutieron y superaron diversas dificultades que, las cuales parecían no poder permitir el avance y conclusión del trabajo, sin embargo, siempre se mantuvo la determinación de concluir el trabajo. La experiencia adquirida es de suma importancia ya que con esta clase de oportunidades es donde realmente el estudiante tiene la capacidad de innovar, experimentar e inventar desde su propia perspectiva, permitiendo así que se dé la chispa del conocimiento, adquiriendo nuevas habilidades y permitiendo ir más allá de la imaginación. Durante el desarrollo de este video juego se logró aprender habilidades nunca desarrolladas, en especial el manejo de código, de una simple operación aritmética a desarrollar un sistema complejo usando recursividad, fue todo un reto que sin duda dejó mucho que aprender.

Siempre hay áreas en constante mejora, sería bueno destacar que muchas partes de este proyecto se pueden mejorar, por ejemplo; hacer que los enemigos se generen de manera aleatoria o que los bloques se vuelvan a dibujar trans cada nivel.

Finalmente se concluye el desarrollo de Vintage Bomberman.

Bibliografía

Programación ATS. “Si puedes imaginarlo, puedes programarlo”. Recuperado de

https://youtube.com/@ProgramacionATS?si=_Jla-4F_hzSrsKVp

Imágenes para la animación de la bomba:

<https://opengameart.org/content/bomb-party-expansion>

Sprite de la skin #1 Bomberman:

https://tcrf.net/Neo_Bomberman/Unused_Graphics

Sprite de la puerta que sube un nivel

<https://www.pinterest.ca/pin/door-with-arch--701717185705848498/>

Tutoriales de Gimp:

https://youtu.be/MiFkQhvimIY?si=y1hs-4_HHXem0NpC

Como hacer que Python cree, lee y edite archivos de texto

https://www.w3schools.com/python/python_file_write.asp

Fondo de la pantalla información:

<https://wallpapersafari.com/bomberman-backgrounds/>

Imágenes para los diferentes elementos, vida, puntos, etc....

https://www.freepik.com/free-vector/hand-drawn-video-game-set-elements_40484534.htm#query=xp%20points&position=0&from_view=keyword&track=ais&uuid=e9098a52-776e-4d0b-9ad7-651fb7b33488

Imágenes de los bombermans en los lados del menú principal:

<https://images.app.goo.gl/JLvhpUYpf5Dw6MDp6>

Imagen y título del menú:

<https://www.google.co.cr/url?sa=i&url=https%3A%2F%2Fm.youtube.com%2Fwatch%3Fv%3DdraxnAXjhNWM&psig=AOvVaw1qsiKADZNylvYXP-w6QVD&ust=1713141715902000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCPDSqLm8wIUdFQAAAAAdAAAAABAJ>

Se utilizó Krita y Gimp para editar las diferentes imágenes:

<https://krita.org/es/>

<https://www.gimp.org/>

Como recortar imágenes en Gimp:

https://youtu.be/MiFkQhvimIY?si=y1hs-4_HHXem0NpC

Curso de pygame:

https://youtu.be/5v_Jl6tMU68?si=V63mdywuiP42PJ8h

Documentación oficial de pygame:

<https://www.pygame.org/docs/>

Creación de menús:

<https://youtu.be/GMBqjxcKogA?si=2nJb3QmHROFdKeoq>

Movimiento de objetos en pygame:

<https://youtu.be/ZcWwgeLp0T8?si=kf3DDAlfTZsNCXUX>

Colores para las Paletas de colores:

<https://htmlcolorcodes.com/>

Música de fondo:

<https://youtu.be/l5D8xW69t8U?si=cirbigpOsSD1ZOJh>

Movimiento del personaje:

<https://youtu.be/BN3kff8gGYA?si=jTrMciBCR0T8jfj0>