Videojuego en Python

Presentado por:

Daniel Felipe Mejia Rios

Profesor/a:

Nestor Dario Duque Mendez

Fecha

14 de Noviembre

Universidad Nacional de Colombia

Facultad de administración

2024

CONTENIDO

1. Punto taller

Ingeniería de software – LLM.

Basándose en la presentación entregada y expuesta en sesión anterior, seleccione una de las herramientas presentadas o realice búsqueda en https://theresanaiforthat.com/s/software/ , https://www.futuretools.io/ , TopAI.tools , etc. para escoger una diferente, según la fase de:

- Definir el(los) componente(s) del desarrollo de software en que se enfoca el taller: ingeniería de requisitos; diseño y la arquitectura; desarrollo de código; aseguramiento de la calidad; documentación de software.
- Según el interés seleccionar la o las herramientas que se adecuen) a los resultados esperados. Ejecutar los diferentes pasos. Probar los resultados arrojados, evaluar y refinar hasta que se obtenga lo deseado.
- Hacer una guía paso a paso sobre el procedimiento utilizado (incluyendo las instalaciones y configuraciones que se requieran) y los resultados obtenidos.
- Subir el documento editable y el código generado.
- Se hará una presentación en clase.

La calificación será **directamente proporcional** a la magnitud de la situación enfrentada, el número de componentes involucrados, la calidad de la salida y la relevancia del informe presentado.

2. Ingeniería de software (Videojuego en consola de comandos)

2.1. **Ingeniería de Requisitos**: Definiremos los requisitos del juego simple que vamos a desarrollar, incluyendo las funcionalidades básicas y las expectativas de los usuarios.

los requisitos básicos del juego son:

- Visual Studio Code: como ambiente de desarrollo para permitir correr el juego
- Entrada de teclado para usar las funciones de movimiento del personaje
- Sistema Operativo compatible con el lenguaje de programación

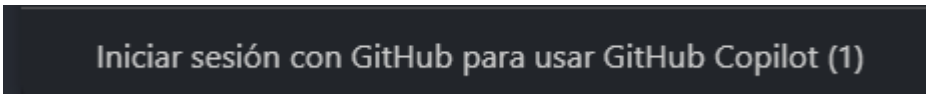
2.2. **Diseño y Arquitectura**: el juego consiste en un personaje hecho a palos tratando de alcanzar una meta por medio de 3 plataformas en distintas posiciones, al alcanzar la meta se reproducirá un sonido que notifique la victoria del jugador y reiniciando el nivel con diferentes posiciones de las plataformas de forma aleatoria, al igual que la meta

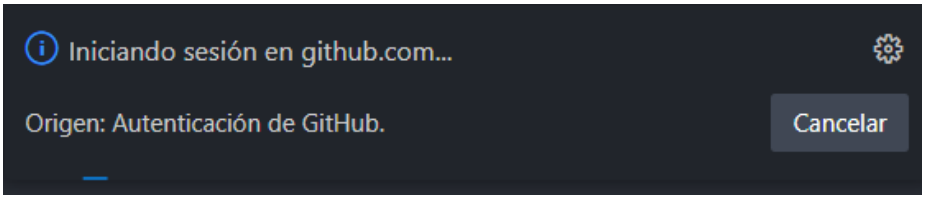
2.3. **Desarrollo de Código**: Implementaremos el juego en Python utilizando la biblioteca pygame, siguiendo las mejores prácticas de programación.

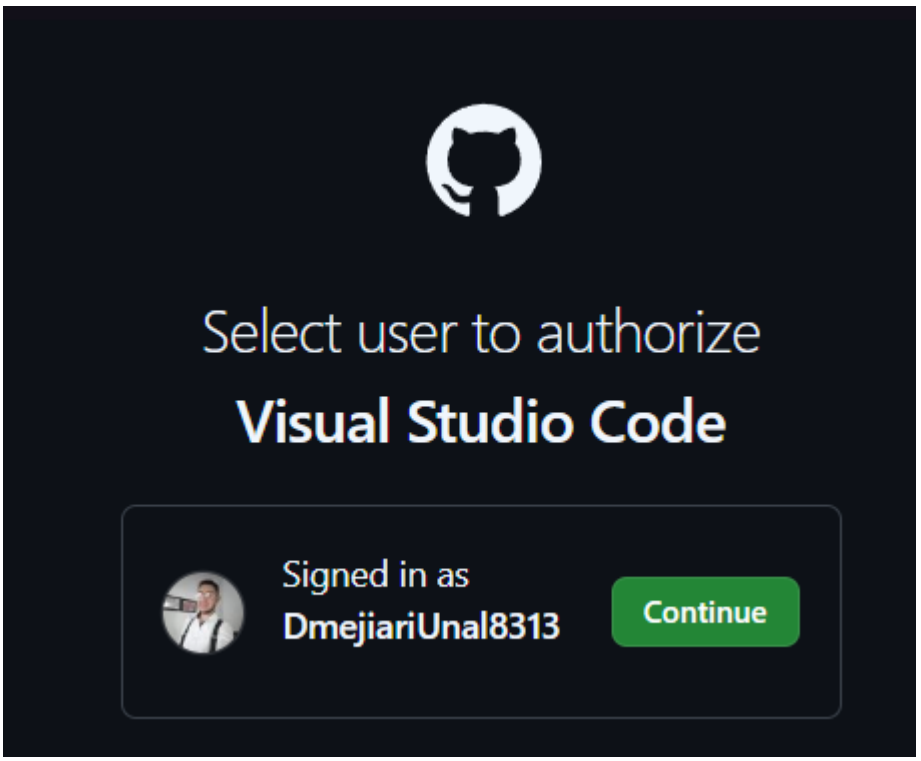
2.4. **Aseguramiento de la Calidad**: Durante el proceso de desarrollo en conjunto a la herramienta Github Copilot, se realizaron pruebas de evaluación y debug que mostraban el proceso del pequeño aplicativo en Python, es importante resaltar el paso a paso de cómo integrar Github Copilot a Visual Studio Code, estos son los pasos a seguir:

2.4.1. Instalar Extension GitHub Copilot y GitHub Copilot Chat desde la sección de Extensiones

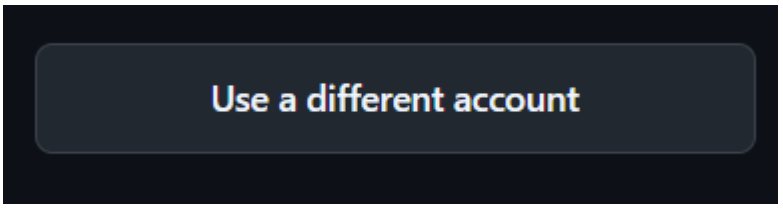
2.4.2. Iniciar sesión de GitHub en Visual Studio Code para trabajar con todos las extensiones que ofrece GitHub, en especial con Copilot para recomendar código sobre el editor y Copilot Chat soportado por GPT 4o



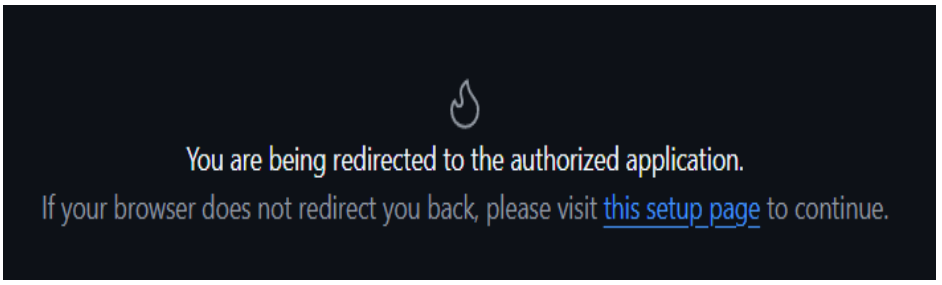




puede iniciar sesion con la cuenta registrada en su navegador predeterminado o escoger otra cuenta diferente, es de tener en cuenta que la cuenta que se escoja debe contar con la suscripción de GitHub Copilot, ya sea personal o adquirida por medio del programa GitHub Education

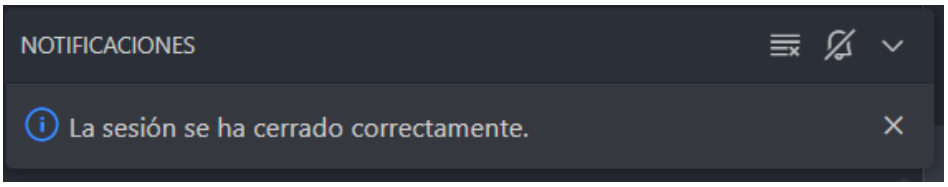


2.4.3. Autenticación del sistema



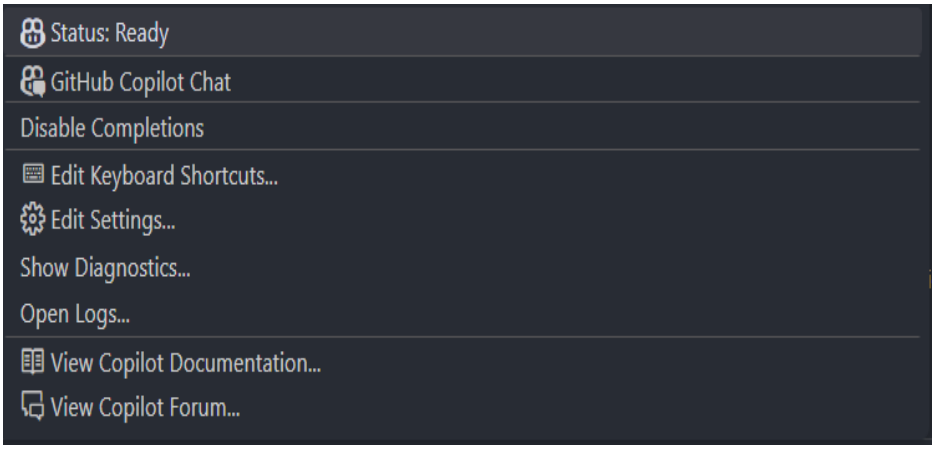
luego de la autenticación, la página se conectara con Visual Studio Code y

activará la extensión

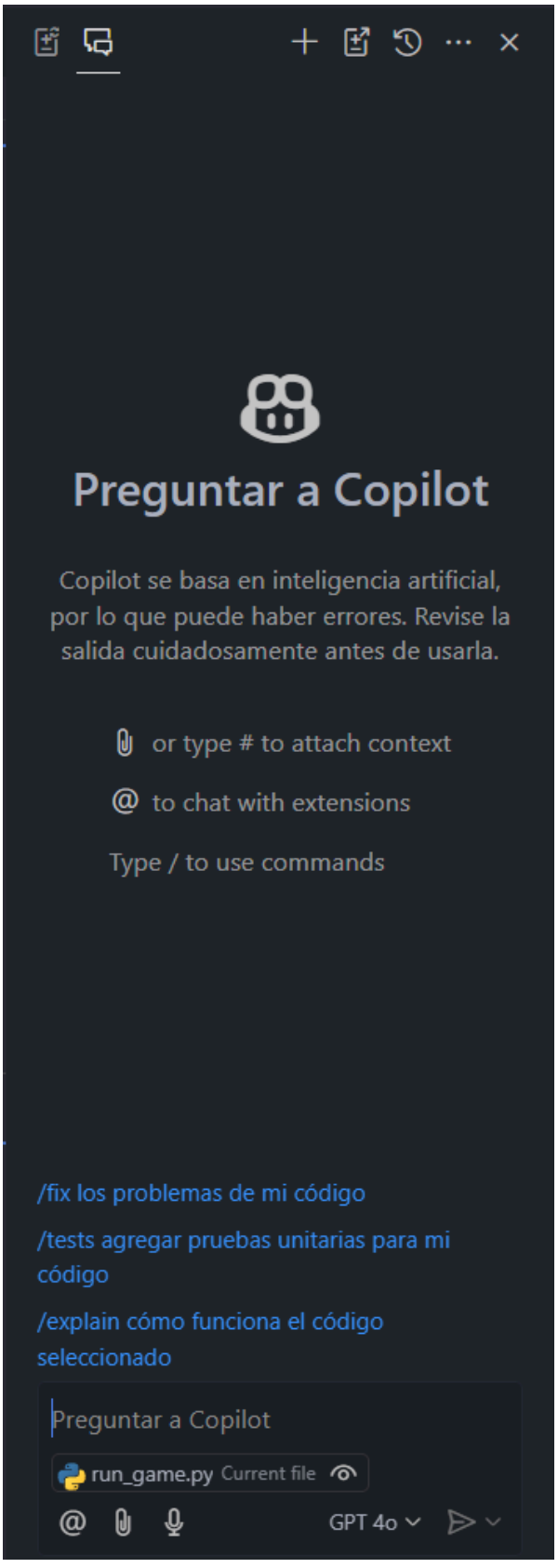


2.4.4. Verificar estado de la extensión

 Una forma de ver el estado de la extensión es notar el símbolo de la misma en el tablero del aplicativo, una vez se presione sobre el permitirá ver el estado y las configuraciones que se le puede aplicar



2.4.5. Prueba de GitHub Copilot Chat



al presionar en el boton github copilot chat se abrirá esta pestaña similar a

un chatbot que entregara recomendaciones de código, ingeniería de software y desarrollo básico sobre uno o varios archivos que se escojan para analizar, un ejemplo de su uso es con este juego cuyo paso a paso será retratado en el siguiente capítulo.

3. Paso a paso

3.1. Instalación de Pygame

1. Instala `pygame`:

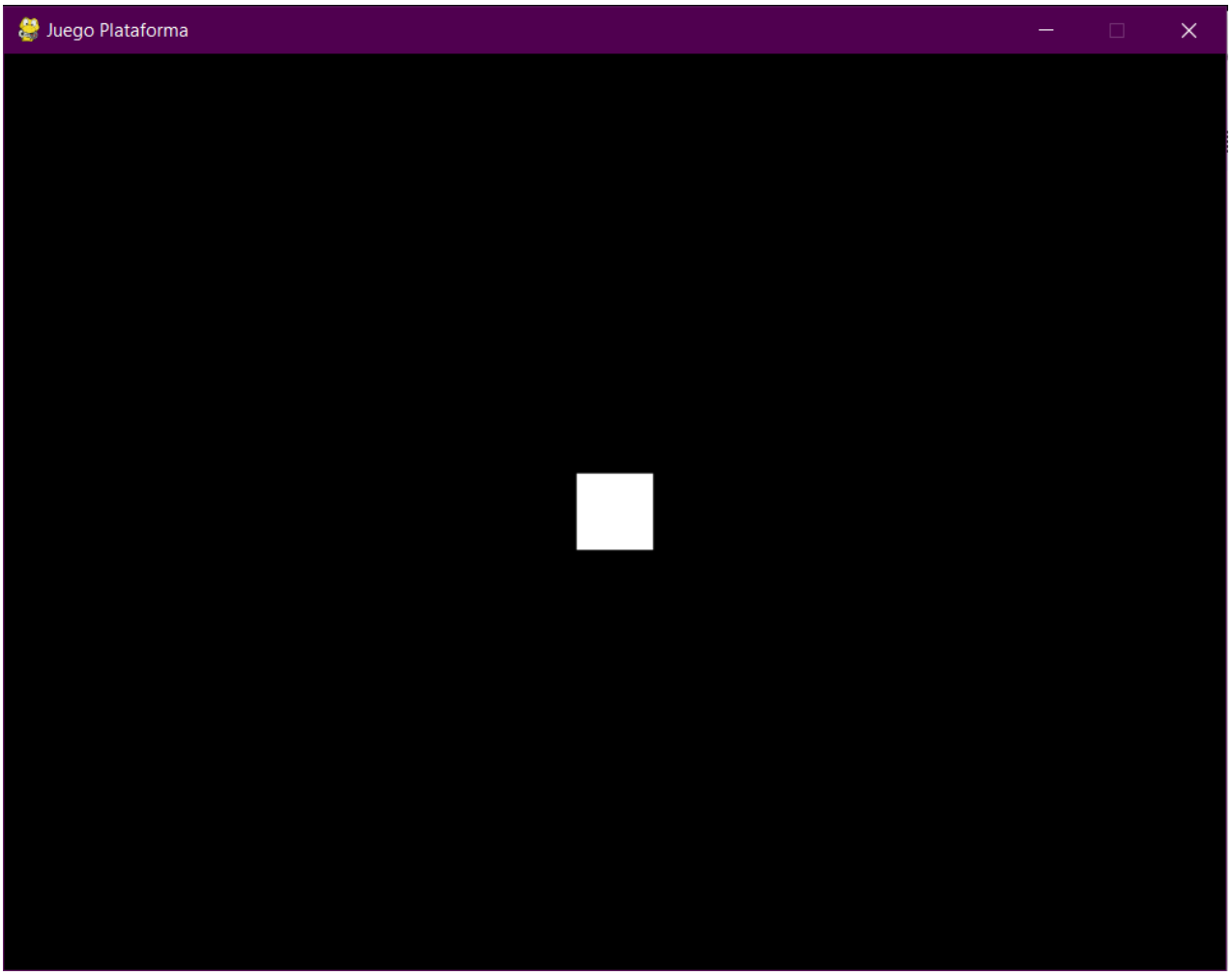
```
pip install pygame
```

Pygame es una biblioteca popular para el desarrollo de videojuegos en Python que proporciona funcionalidades para manejar gráficos, sonido y entradas de usuario.

3.2 Definición de código fuente base

```python
import pygame
import sys

# Inicializar Pygame
pygame.init()

# configurar la pantalla
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Juego Plataforma')

# colores
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)

# posición inicial del jugador
player_pos = [400, 300]
player_size = 50

# velocidad del jugador
player_speed = 5

# bucle principal del juego
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # obtener las teclas presionadas
    keys = pygame.key.get_pressed()

    # mover al jugador
    if keys[pygame.K_LEFT]:
        player_pos[0] -= player_speed
    if keys[pygame.K_RIGHT]:
        player_pos[0] += player_speed
    if keys[pygame.K_UP]:
        player_pos[1] -= player_speed
    if keys[pygame.K_DOWN]:
        player_pos[1] += player_speed

    # rellena la pantalla con negro
    screen.fill(BLACK)

    # Dibuja el jugador
    pygame.draw.rect(screen, WHITE, (player_pos[0], player_pos[1],player_size,player_size))

    # actualiza la pantalla
    pygame.display.flip()

    # controla la velocidad del juego
    pygame.time.Clock().tick(30)
```

3.3 primera prueba



Para este punto después de haber desarrollado el código base y ejecutarlo como prueba inicial, el cuadro se permitirá moverse por toda la pantalla en los 4 ejes (arriba, abajo, izquierda y derecha) de acuerdo los botones que el usuario presiones

3.4 Modificaciones

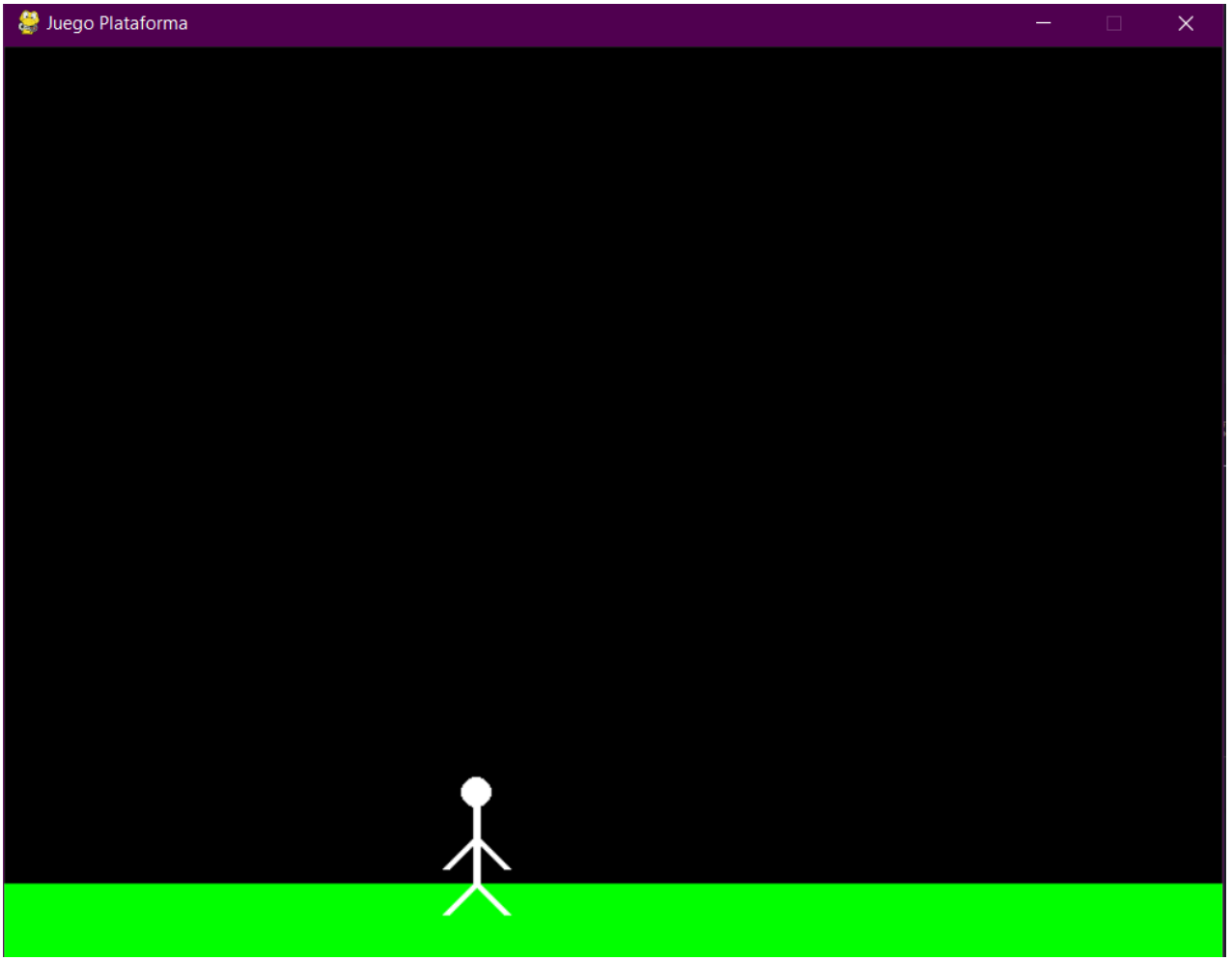
```
# velocidad del jugador
player_speed = 5
jump_speed = 10
gravity = 0.5
is_jumping = False
jump_velocity = 0
```

```
# Superficie plana
ground_y = 550
```

```python
# mover al jugador horizontalmente
if keys[pygame.K_LEFT]:
    player_pos[0] -= player_speed
if keys[pygame.K_RIGHT]:
    player_pos[0] += player_speed

# saltar
if keys[pygame.K_SPACE] and not is_jumping:
    is_jumping = True
    jump_velocity = jump_speed

# aplicar gravedad
if is_jumping:
    player_pos[1] -= jump_velocity
    jump_velocity -= gravity
    if player_pos[1] >= ground_y - player_size:
        player_pos[1] = ground_y - player_size
        is_jumping = False
```

```python
# Dibuja la superficie plana
pygame.draw.rect(screen, WHITE, (0, ground_y, 800, 50))
```

de acuerdo a algunas pruebas el personaje se quedaba suspendido en el espacio o no saltaba correctamente, asi que los pasos que se sugirieron fueron modificar la logica de salto y el posicionamiento del personaje dando como resultado la siguiente prueba:



estos cambios se dieron en los siguientes fragmentos del código

```
# Aplicar gravedad
if is_jumping:
    player_pos[1] += jump_velocity
    jump_velocity += gravity
    if player_pos[1] >= ground_y - player_size:
        player_pos[1] = ground_y - player_size
        is_jumping = False
        jump_velocity = 0
```

```
# Superficie plana
ground_y = 550

# Posición inicial del jugador
player_pos = [400, ground_y - 50]
player_size = 50
```

3.5 Modificaciones fuertes

ahora el sistema para estar mejor definido, se ha creado un sistema de plataformas y una meta por alcanzar, reproduciendo un sonido cada que logra alcanzar la meta, estos son los codigos usados

```
# Meta
goal_pos = [750, ground_y - 50]
goal_size = 50

# Plataformas
platforms = [
    pygame.Rect(200, 450, 100, 10),
    pygame.Rect(400, 350, 100, 10),
    pygame.Rect(600, 250, 100, 10)
]

# Cargar sonido de victoria
victory_sound = pygame.mixer.Sound('victory.wav')

def reset_level():
    global player_pos, is_jumping, jump_velocity
    player_pos = [400, ground_y - 50]
    is_jumping = False
    jump_velocity = 0
    randomize_platforms()
```

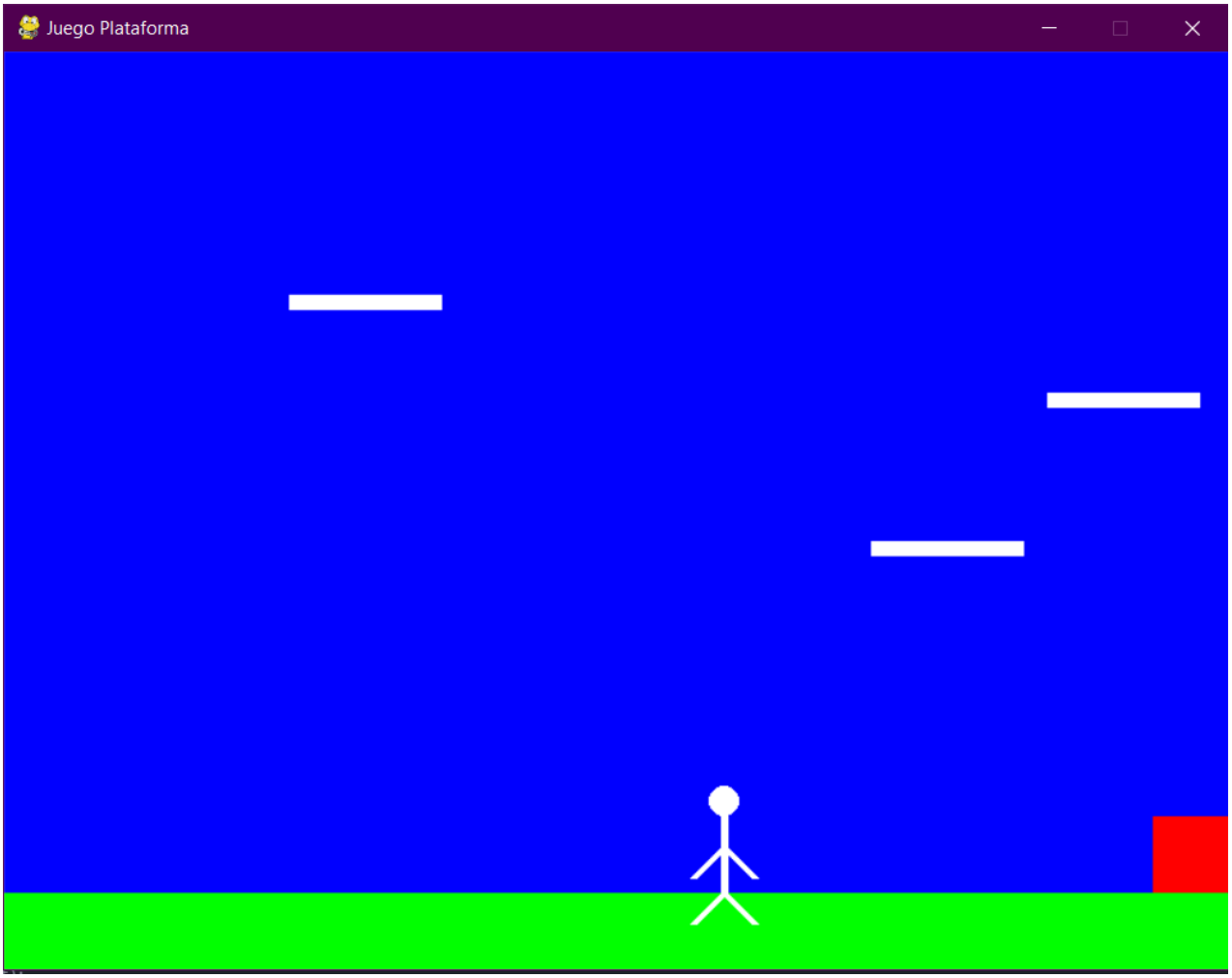
```
def randomize_platforms():
    for platform in platforms:
        platform.x = random.randint(0, 700)
        platform.y = random.randint(100, 500)
```

```python
# Colisiones con plataformas
player_rect = pygame.Rect(player_pos[0], player_pos[1], player_size, player_size)
for platform in platforms:
    if player_rect.colliderect(platform) and jump_velocity > 0:
        player_pos[1] = platform.y - player_size
        is_jumping = False
        jump_velocity = 0

# Verificar si el jugador ha llegado a la meta
goal_rect = pygame.Rect(goal_pos[0], goal_pos[1], goal_size, goal_size)
if player_rect.colliderect(goal_rect):
    victory_sound.play()
    reset_level()
```

```python
# Dibuja la meta
pygame.draw.rect(screen, RED, goal_rect)

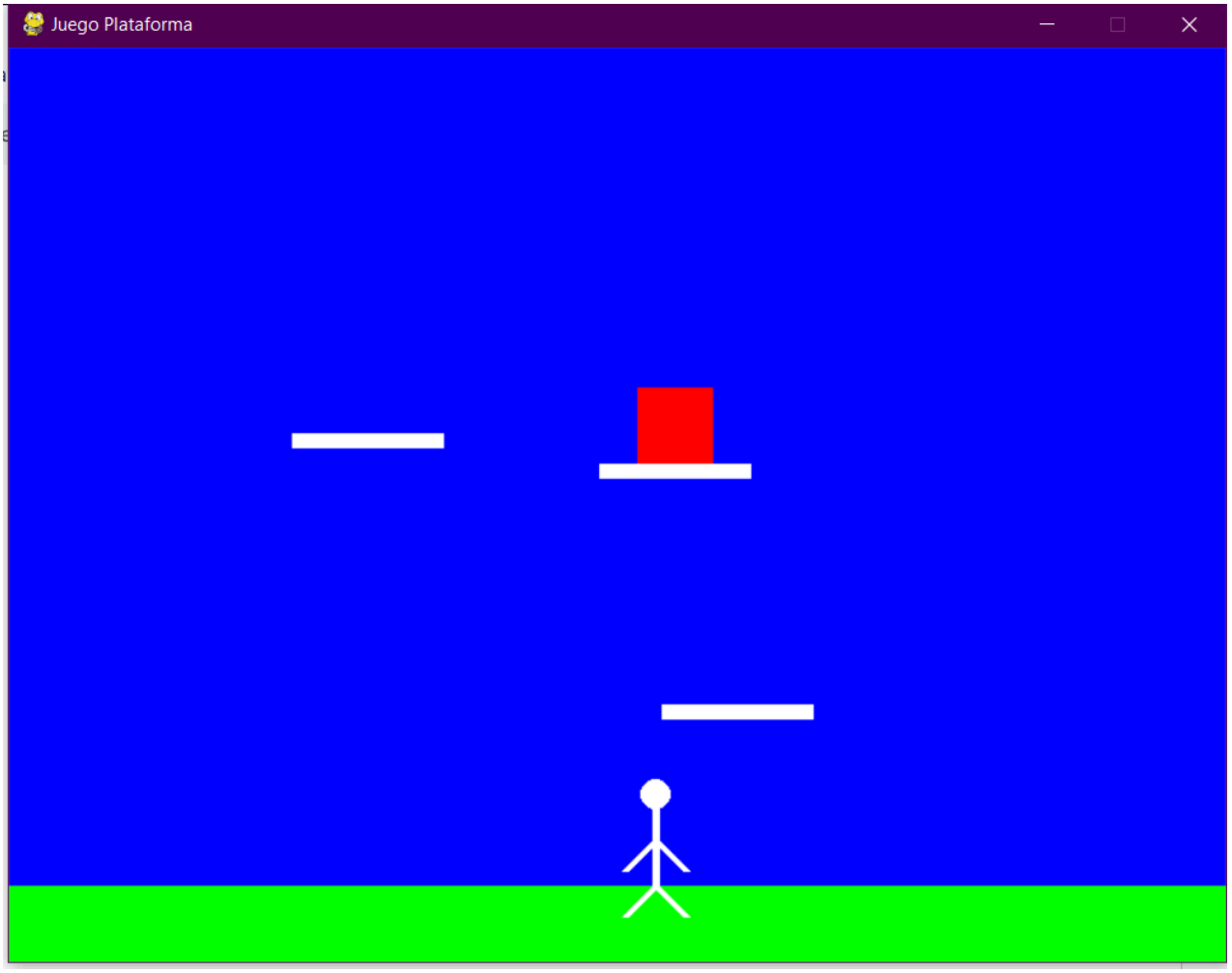
# Dibuja las plataformas
for platform in platforms:
    pygame.draw.rect(screen, WHITE, platform)
```

dando como resultado la siguiente prueba



3.6 Paso final

Después de algunas últimas pruebas, el juego está en su mayor punto, desde que era un cuadro hasta una persona en una superficie y ahora en plataforma hasta una meta y cada que cumple esa meta el juego hará un sonido de victoria, siendo infinito en la generación de niveles



4. Resultados

4.1 Codigo fuente

```
import pygame
import sys
import random

# Inicializar Pygame
pygame.init()

# Configurar la pantalla
```

```
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Juego Plataforma')

# Colores
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
RED = (255, 0, 0)

# Superficie plana
ground_y = 550

# Posición inicial del jugador
player_pos = [400, ground_y - 50]
player_size = 50

# Velocidad del jugador
player_speed = 5
jump_speed = 10
gravity = 0.5
is_jumping = False
jump_velocity = 0

# Meta
goal_size = 50
goal_pos = [0, 0]

# Plataformas
platforms = [
    pygame.Rect(200, 450, 100, 10),
    pygame.Rect(400, 350, 100, 10),
    pygame.Rect(600, 250, 100, 10)
]

# Cargar sonido de victoria
victory_sound = pygame.mixer.Sound('victory.wav')

def reset_level():
    global player_pos, is_jumping, jump_velocity, goal_pos
```

```
    player_pos = [400, ground_y - 50]
    is_jumping = False
    jump_velocity = 0
    randomize_platforms()
    place_goal()

def randomize_platforms():
    for platform in platforms:
        platform.x = random.randint(0, 700)
        platform.y = random.randint(100, 500)

def place_goal():
    platform = random.choice(platforms)
    goal_pos[0] = platform.x + (platform.width - goal_size) // 2
    goal_pos[1] = platform.y - goal_size

# Inicializar el primer nivel
reset_level()

# Bucle principal del juego
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Obtener las teclas presionadas
    keys = pygame.key.get_pressed()

    # Mover al jugador horizontalmente
    if keys[pygame.K_LEFT]:
        player_pos[0] -= player_speed
    if keys[pygame.K_RIGHT]:
        player_pos[0] += player_speed

    # Saltar
    if keys[pygame.K_SPACE]:
        jump_velocity = -jump_speed
        is_jumping = True
```

```
    # Aplicar gravedad
    if is_jumping:
        player_pos[1] += jump_velocity
        jump_velocity += gravity
        if player_pos[1] >= ground_y - player_size:
            player_pos[1] = ground_y - player_size
            is_jumping = False
            jump_velocity = 0

    # Colisiones con plataformas
    player_rect = pygame.Rect(player_pos[0], player_pos[1], player_size,
player_size)
    for platform in platforms:
        if player_rect.colliderect(platform) and jump_velocity > 0:
            player_pos[1] = platform.y - player_size
            is_jumping = False
            jump_velocity = 0

    # Verificar si el jugador ha llegado a la meta
    goal_rect = pygame.Rect(goal_pos[0], goal_pos[1], goal_size, goal_size)
    if player_rect.colliderect(goal_rect):
        victory_sound.play()
        reset_level()

    # Rellena la pantalla con el color de fondo
    screen.fill(BLUE)

    # Dibuja la superficie plana
    pygame.draw.rect(screen, GREEN, (0, ground_y, 800, 50))

    # Dibuja la meta
    pygame.draw.rect(screen, RED, goal_rect)

    # Dibuja las plataformas
    for platform in platforms:
        pygame.draw.rect(screen, WHITE, platform)

    # Dibuja la persona hecha a palos
    # Cuerpo
```

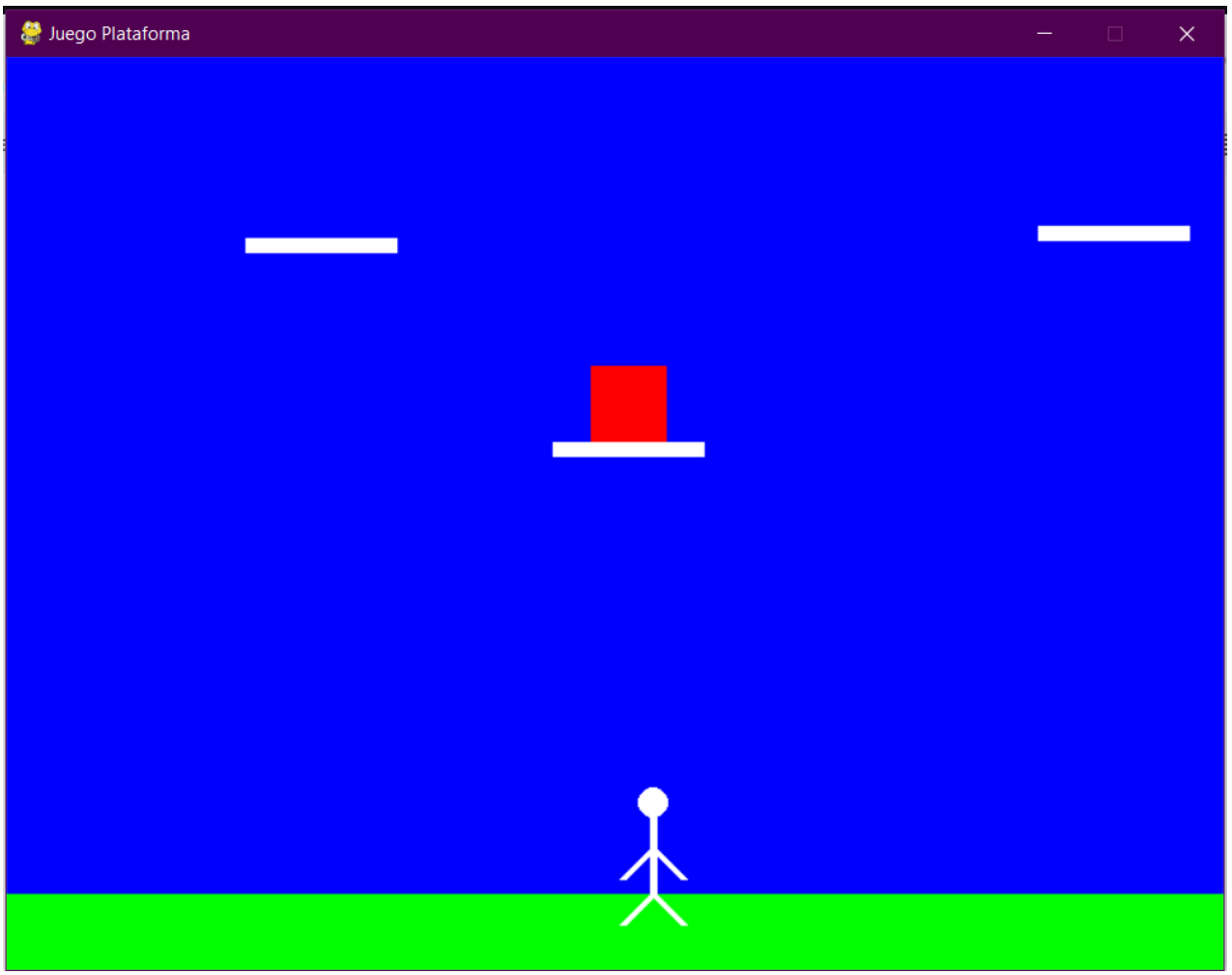
```
    pygame.draw.line(screen, WHITE, (player_pos[0] + player_size // 2,
player_pos[1]), (player_pos[0] + player_size // 2, player_pos[1] +
player_size), 5)
    # Cabeza
    pygame.draw.circle(screen, WHITE, (player_pos[0] + player_size // 2,
player_pos[1] - 10), 10)
    # Brazos
    pygame.draw.line(screen, WHITE, (player_pos[0] + player_size // 2,
player_pos[1] + 20), (player_pos[0] + player_size // 2 - 20, player_pos[1]
+ 40), 5)
    pygame.draw.line(screen, WHITE, (player_pos[0] + player_size // 2,
player_pos[1] + 20), (player_pos[0] + player_size // 2 + 20, player_pos[1]
+ 40), 5)
    # Piernas
    pygame.draw.line(screen, WHITE, (player_pos[0] + player_size // 2,
player_pos[1] + player_size), (player_pos[0] + player_size // 2 - 20,
player_pos[1] + player_size + 20), 5)
    pygame.draw.line(screen, WHITE, (player_pos[0] + player_size // 2,
player_pos[1] + player_size), (player_pos[0] + player_size // 2 + 20,
player_pos[1] + player_size + 20), 5)

    # Actualiza la pantalla
    pygame.display.flip()

    # Controla la velocidad del juego
    pygame.time.Clock().tick(30)

pygame.quit()
sys.exit()
```

juego



5. Como jugar al juego

Objetivo del Juego

El objetivo del juego es mover al personaje a través de las plataformas y alcanzar la meta (representada por un rectángulo rojo) en cada nivel. Al alcanzar la meta, se reproducirá un sonido de victoria y se generará un nuevo nivel con plataformas y una meta en nuevas posiciones aleatorias.

Controles del Juego

- Mover a la Izquierda: Presiona la tecla de flecha izquierda (←) para mover al personaje hacia la izquierda.
- Mover a la Derecha: Presiona la tecla de flecha derecha (→) para mover al personaje hacia la derecha.

- Saltar: Presiona la barra espaciadora (Space) para hacer que el personaje salte. El personaje puede realizar hasta dos saltos consecutivos antes de aterrizar en una plataforma o en el suelo.

Mecánicas del Juego

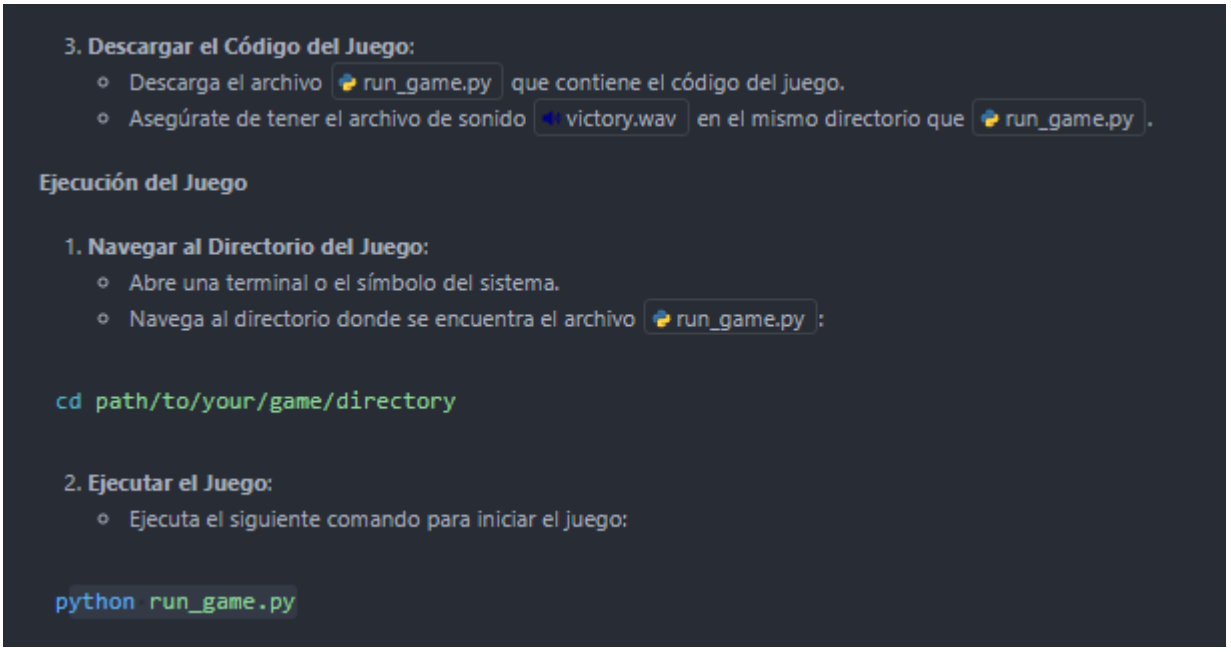
1. Movimiento Horizontal:
    - Usa las teclas de flecha izquierda y derecha para mover al personaje horizontalmente a través de la pantalla.
2. Salto y Doble Salto:
    - Presiona la barra espaciadora para hacer que el personaje salte.
    - El personaje puede realizar un segundo salto mientras está en el aire, permitiendo alcanzar plataformas más altas.
    - El contador de saltos se restablece cuando el personaje aterriza en el suelo o en una plataforma.
3. Interacción con Plataformas:
    - El personaje puede aterrizar en plataformas distribuidas aleatoriamente en el nivel.
    - Las plataformas permiten al personaje moverse a diferentes alturas y alcanzar la meta.
4. Meta:
    - La meta está representada por un rectángulo rojo colocado aleatoriamente en una de las plataformas.
    - Al alcanzar la meta, se reproducirá un sonido de victoria y se generará un nuevo nivel.

6. Herramienta usada

GitHub Copilot

7. Pasos de instalación

- Instalar Python:
- Descarga e instala Python desde python.org.
- Asegúrate de agregar Python al PATH durante la instalación.
- Instalar Pygame:

- Abre una terminal o el símbolo del sistema.
- Ejecuta el siguiente comando para instalar pygame:
- pip install pygame
- ```
  pip install pygame
  ```
- 

  ```