

Universidad Nacional de Costa Rica

Integrantes:

Justin Andrés Briones Sequeira

Emmanuel Rodriguez Picado

Josué Vargas Gutierrez

Carrera:

Ingeniería en Sistemas para la información

Curso:

Estructura de datos

Docente:

José Pablo Calvo Suarez

Proyecto 1:

Courier Quest - Bitácora

## Bitácora de prompts

Día 1: “Estoy desarrollando un juego en PyCharm con pygame y tengo que hacer una cuadrícula con edificios, un jugador y objetos que se deben recoger y entregar en diferentes posiciones, eso viene de una API, como extraigo el contenido de dicha API, que librería debería de usar y dame un ejemplo.”

“Por otro lado ya hice una clase jugador con sus atributos necesarios, pero no se programar su movimiento, ¿me explicas como hacerlo?”

Código dado: class Jugador:

```
def __init__(self, x, y):  
  
    self.x = x # posición en la cuadrícula (columna)  
  
    self.y = y # posición en la cuadrícula (fila)
```

```
# Método para mover al jugador
```

```
def mover(self, direccion):  
  
    if direccion == "arriba":  
  
        self.y -= 1  
  
    elif direccion == "abajo":  
  
        self.y += 1  
  
    elif direccion == "izquierda":  
  
        self.x -= 1  
  
    elif direccion == "derecha":  
  
        self.x += 1
```

```
import pygame
```

```
pygame.init()
```

```
pantalla = pygame.display.set_mode((500, 500))
```

```
jugador = Jugador(5, 5) # posición inicial
```

```
TAM_CELDA = 50 # tamaño de cada celda de la cuadrícula
```

```
corriendo = True
```

```
while corriendo:
```

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

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

```
            corriendo = False
```

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

```
            if evento.key == pygame.K_w: # W -> arriba
```

```
                jugador.mover("arriba")
```

```
            elif evento.key == pygame.K_s: # S -> abajo
```

```
                jugador.mover("abajo")
```

```
            elif evento.key == pygame.K_a: # A -> izquierda
```

```
                jugador.mover("izquierda")
```

```
            elif evento.key == pygame.K_d: # D -> derecha
```

```
                jugador.mover("derecha")
```

```
        # Dibujar
```

```
        pantalla.fill((0, 0, 0)) # limpiar pantalla
```

```
        pygame.draw.rect(pantalla, (0, 255, 0),
```

```
                           (jugador.x*TAM_CELDA, jugador.y*TAM_CELDA, TAM_CELDA,
                           TAM_CELDA))
```

```
        pygame.display.flip()
```

Día 2: “Hola, ya programé el movimiento del jugador y también tengo ya la extracción de pedidos de la API, ¿ahora me puedes explicar cómo dibujar las cosas obtenidas de la API? Por ahora solo sería pedidos, jugador y mapa”

Ya para este punto ya se podía mover al jugador y también se podía ver reflejados los pedidos ya dibujados en el juego y también el mapa, faltaba la implementación de recoger y “dropear” los pedidos.

Código dado:

### ❶ Supongamos tus datos

- **Mapa:** podrías tener una cuadrícula de tamaño fijo, por ejemplo 10x10 celdas.
- **Jugador:** tiene x y y en la cuadrícula.
- **Pedidos:** lista de objetos con coordenadas y tipo, por ejemplo:

```
pedidos = [  
    {"x": 1, "y": 2, "tipo": "paquete"},  
    {"x": 4, "y": 5, "tipo": "caja"}  
]
```

---

### ❷ Definir tamaños y colores

```
TAM_CELDA = 50 # ancho y alto de cada celda  
COLOR_CELDA = (200, 200, 200)  
COLOR_JUGADOR = (0, 255, 0)  
COLOR_PEDIDO = (255, 0, 0)
```

---

### ❸ Dibujar la cuadrícula

```
def dibujar_mapa(pantalla, filas, columnas):  
    for y in range(filas):  
        for x in range(columnas):  
            pygame.draw.rect(pantalla, COLOR_CELDA,  
                             (x*TAM_CELDA, y*TAM_CELDA, TAM_CELDA, TAM_CELDA), 1) # 1  
= borde
```

---

### ❹ Dibujar al jugador

```
def dibujar_jugador(pantalla, jugador):  
    pygame.draw.rect(pantalla, COLOR_JUGADOR,  
                      (jugador.x*TAM_CELDA, jugador.y*TAM_CELDA, TAM_CELDA,  
                       TAM_CELDA))
```

---

## **5 Dibujar los pedidos**

```
def dibujar_pedidos(pantalla, pedidos):  
    for pedido in pedidos:  
        pygame.draw.rect(pantalla, COLOR_PEDIDO,  
                          (pedido["x"]*TAM_CELDA, pedido["y"]*TAM_CELDA, TAM_CELDA,  
                           TAM_CELDA))
```

---

## **6 Integrarlo todo en el bucle del juego**

```
import pygame  
  
pygame.init()  
  
filas, columnas = 10, 10  
  
pantalla = pygame.display.set_mode((columnas*TAM_CELDA, filas*TAM_CELDA))  
  
jugador = Jugador(0, 0) # ejemplo  
pedidos = [{"x": 1, "y": 2}, {"x": 4, "y": 5}] # ejemplo  
  
corriendo = True  
  
while corriendo:  
    for evento in pygame.event.get():  
        if evento.type == pygame.QUIT:  
            corriendo = False
```

```
pantalla.fill((0, 0, 0))          # limpiar pantalla
```

```
dibujar_mapa(pantalla, filas, columnas)
```

```
dibujar_jugador(pantalla, jugador)
```

```
dibujar_pedidos(pantalla, pedidos)
```

```
pygame.display.flip()
```

---

✅ Con esto verías **la cuadrícula, al jugador y los pedidos de la API** en pantalla.

Día 3: “Hola, los pedidos de la API aparecen dentro de los edificios, ¿me ayudarías a implementar un método como `reubicar_pedidos` que saque los pedidos del edificio?”

Código dado:

```
import random
```

```
def reubicar_pedidos(pedidos, mapa, ocupadas=None, separacion=4):
```

```
    """
```

```
    Reubica pedidos para que no queden dentro de edificios
```

```
    ni en casillas ocupadas, con separación mínima.
```

```
    """
```

```
    if ocupadas is None:
```

```
        ocupadas = set()
```

```
    filas = len(mapa)
```

```
    columnas = len(mapa[0])
```

```

for p in pedidos:

for punto in ["pickup", "dropoff"]:

x0, y0 = p[punto]


if mapa[y0][x0] == "B" or (x0, y0) in ocupadas:

    # Intentos de ubicar alrededor en un área de 7x7 por defecto

    posibles = [

        (x0+dx, y0+dy)

        for dx in range(-separacion, separacion+1)

        for dy in range(-separacion, separacion+1)

        if 0 <= x0+dx < columnas and 0 <= y0+dy < filas

        and mapa[y0+dy][x0+dx] != "B"

        and (x0+dx, y0+dy) not in ocupadas

    ]

    if posibles:

        nx, ny = random.choice(posibles)

        p[punto] = [nx, ny]

        ocupadas.add((nx, ny))

    else:

        # Si no hay opción libre, dejar donde está

        ocupadas.add((x0, y0))

else:

    ocupadas.add((x0, y0))

```

Día 4: “Hola, estoy haciendo un proyecto en Python con Pycharm, podrías decirme cómo puedo ponerle una imagen a el jugador?”

Código dado:

¡Hola! Claro que sí. Si estás haciendo un proyecto en Python (usando PyCharm o IntelliJ IDEA con el plugin de Python), y quieres **ponerle una imagen a un jugador**, lo más común es que estés usando alguna **librería gráfica** como:

- `pygame` (la más usada para juegos 2D)
- `tkinter` (para interfaces gráficas básicas)
- `arcade`, `pyglet`, etc.

✅ Supondré que estás usando `pygame`, que es lo más común para juegos.

Aquí te explico cómo ponerle una imagen a un jugador en `pygame`, paso a paso:

---

## Pasos para poner una imagen al jugador con `pygame`

### 1. Instala `pygame` (si no lo tienes):

```
pip install pygame
```

### 2. Estructura básica del código:

```
import pygame
```

```
import sys
```

```
# Inicializar pygame
```

```
pygame.init()
```

```
# Crear ventana
```

```
screen = pygame.display.set_mode((800, 600))
```



```
pygame.display.set_caption("Mi Juego")

# Cargar imagen del jugador

jugador_imagen = pygame.image.load("jugador.png") # Asegúrate de
que el archivo esté en el mismo directorio

# Posición inicial del jugador

jugador_x = 100

jugador_y = 300

# Bucle principal del juego

running = True

while running:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False

    # Fondo blanco

    screen.fill((255, 255, 255))

    # Dibujar jugador

    screen.blit(jugador_imagen, (jugador_x, jugador_y))

    # Actualizar pantalla

    pygame.display.flip()
```

```
# Salir

pygame.quit()

sys.exit()
```

---

## Notas importantes:

- Guarda tu imagen (por ejemplo `jugador.png`) en la misma carpeta donde está tu archivo `.py`, o usa la ruta completa.
- La imagen **debe ser compatible con pygame**: PNG o JPG, y no muy grande (idealmente < 512x512 px).
- Puedes usar `pygame.transform.scale()` si quieres redimensionar la imagen:

```
jugador_imagen = pygame.transform.scale(jugador_imagen, (64, 64)) #  
Cambia a 64x64 px
```

Día 5: “Ocupo que los pedidos sean siempre diferentes, que puedo hacer, que no se copien ? ”

Para asegurarte de que los pedidos siempre sean diferentes y no se copien, puedes usar un `set()` para llevar control de las posiciones ya ocupadas.

La idea es que cada vez que vayas a asignar o reubicar un pedido, verifiques si la posición ya está en el conjunto `ocupadas`. Si no lo está, lo agregas al set; si ya está, buscas otra posición libre.

Por ejemplo:

```
ocupadas = set() # almacena tuplas (x, y) de posiciones ocupadas
```

```
for pedido in pedidos:
```

```
    x, y = pedido["pickup"] # ejemplo para pickup
```

```
    if (x, y) not in ocupadas:
```

```
        ocupadas.add((x, y)) # marcar como ocupada
```

```
    else:
```

```
        # buscar nueva posición libre
```

```
        for dx in range(-1, 2):
```

```
            for dy in range(-1, 2):
```

```
                nx, ny = x + dx, y + dy
```

```
                if (nx, ny) not in ocupadas:
```

```
                    pedido["pickup"] = [nx, ny]
```

```
                    ocupadas.add((nx, ny))
```

```
                    break
```

```
            else:
```

```
                continue
```

break

De esta manera, **ningún pedido ocupará la misma celda que otro**, y siempre tendrás pedidos diferentes en el mapa.

Día 6: “Hola, estoy haciendo un proyecto en Python con Pycharm, ¿podrías decirme cómo puedo hacer que el juego termine después de una cierta cantidad de tiempo?”

Código dado: ¡Hola! Claro que sí. Si estás haciendo un juego en Python (usando PyCharm o IntelliJ IDEA con soporte Python) y quieres que el juego termine después de un cierto tiempo, lo más común es que estés usando pygame, que tiene funciones muy útiles para manejar el tiempo.

✅ Aquí te explico cómo hacer que el juego termine automáticamente después de X segundos usando pygame.

📌 Ejemplo completo con temporizador:

Este ejemplo hace que el juego termine después de 10 segundos:

```
import pygame
```

```
import sys
```

```
# Inicializar pygame
```

```
pygame.init()
```

```
# Configurar la pantalla
```

```
screen = pygame.display.set_mode((800, 600))
```

```
pygame.display.set_caption("Juego con temporizador")
```

```
# Configurar el reloj
```

```
clock = pygame.time.Clock()

# Tiempo de duración del juego (en milisegundos)
duracion_juego = 10 * 1000 # 10 segundos

# Tiempo en que empezó el juego
tiempo_inicio = pygame.time.get_ticks()

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

# Tiempo actual
tiempo_actual = pygame.time.get_ticks()

# Verificar si se cumplió el tiempo
if tiempo_actual - tiempo_inicio >= duracion_juego:
    print("¡Tiempo terminado!")
    running = False

# Lógica del juego y renderizado
screen.fill((255, 255, 255)) # Fondo blanco
```

```
# Actualizar la pantalla
```

```
pygame.display.flip()
```

```
# Limitar a 60 FPS
```

```
clock.tick(60)
```

```
# Salir del juego
```

```
pygame.quit()
```

```
sys.exit()
```

🔍 ¿Qué hace este código?

Usa `pygame.time.get_ticks()` para obtener el tiempo desde que se inició pygame (en milisegundos).

Guarda el tiempo de inicio (`tiempo_inicio`) justo antes de entrar al bucle principal.

Cada frame compara el tiempo actual con el tiempo de inicio, y si pasaron más de 10 segundos (10,000 ms), termina el juego.

Día 6 entrada 2: “¿Podrías hacer que el juego muestre una pantalla de derrota al acabarse el tiempo?”

Código dado: ¡Por supuesto! Vamos a modificar el ejemplo de `pygame` para que, **cuando se acabe el tiempo**, en lugar de cerrar el juego directamente, se muestre una **pantalla de derrota** (por ejemplo, un mensaje que diga “¡Has perdido!”) durante unos segundos antes de cerrar o volver al menú si quieres.



## Ejemplo completo: Pantalla de derrota tras acabarse el tiempo

```
import pygame

import sys

# Inicializar pygame
pygame.init()

# Configurar la pantalla
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Juego con pantalla de derrota")

# Configurar el reloj
clock = pygame.time.Clock()

# Fuente para mostrar texto
fuente = pygame.font.SysFont(None, 72)

# Tiempo de duración del juego (en milisegundos)
duracion_juego = 10 * 1000 # 10 segundos

# Tiempo en que empezó el juego
tiempo_inicio = pygame.time.get_ticks()

# Estado del juego
running = True
```

```
mostrar_derrota = False

# Bucle principal del juego
while running:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False

    # Tiempo actual

    tiempo_actual = pygame.time.get_ticks()

    # Si no estamos en pantalla de derrota

    if not mostrar_derrota:

        if tiempo_actual - tiempo_inicio >= duracion_juego:

            # Cambiar al estado de derrota

            mostrar_derrota = True

            tiempo_inicio_derrota = pygame.time.get_ticks()

        else:

            # Aquí va la lógica del juego normal

            screen.fill((255, 255, 255)) # Fondo blanco

            # Puedes dibujar al jugador, enemigos, etc. aquí

    else:
```



```
# Mostrar pantalla de derrota

screen.fill((0, 0, 0)) # Fondo negro

texto_derrota = fuente.render("¡Has perdido!", True, (255,
0, 0))

screen.blit(texto_derrota, (200, 250))


# Mostrar pantalla de derrota durante 3 segundos

if pygame.time.get_ticks() - tiempo_inicio_derrota > 3000:

    running = False


# Actualizar la pantalla

pygame.display.flip()


# Limitar a 60 FPS

clock.tick(60)


# Salir del juego

pygame.quit()

sys.exit()
```

---

### Explicación rápida:

- Usa una variable `mostrar_derrota` para cambiar el estado del juego.
- Cuando el tiempo se agota, cambia a la pantalla de derrota.

- En la pantalla de derrota:
    - Se cambia el fondo a negro.
    - Se muestra un texto rojo que dice "¡Has perdido!".
    - Después de 3 segundos, se cierra el juego.
- 

## ¿Qué puedes personalizar?

- Texto de derrota ("¡Has perdido!")
- Duración de la pantalla final (3000 ms = 3 segundos)
- Colores, tamaños, fuentes, o incluso mostrar una imagen en lugar de texto.

Día 8: "Hola, estoy haciendo un proyecto en j idea con Python y Pycharm, podrías decirme cómo puedo poner que en la pantalla aparezca una barra con la resistencia que tiene el personaje?"

Código dado: ¡Hola! Sí, claro que puedo ayudarte. Estás haciendo un proyecto en **Python**, probablemente con **pygame**, y quieres mostrar una **barra de resistencia (stamina)** para tu personaje. Esta es una característica muy común en juegos, y se puede implementar fácilmente con **pygame**.

---

## ¿Qué es una barra de resistencia?

Una **barra de resistencia** (o stamina) suele ser una barra que:

- **Disminuye** cuando el personaje corre, salta, ataca, etc.
  - **Se regenera** lentamente con el tiempo.
  - Se muestra en la pantalla como una **barra horizontal**, usualmente de color verde o amarillo.
- 

## Ejemplo de código: barra de resistencia en pygame

Este ejemplo muestra una barra de resistencia que puedes controlar y regenerar:

```
import pygame

import sys


# Inicializar pygame

pygame.init()


# Configurar la pantalla

screen = pygame.display.set_mode((800, 600))

pygame.display.set_caption("Juego con barra de resistencia")


# Reloj para FPS

clock = pygame.time.Clock()


# Colores

COLOR_FONDO = (30, 30, 30)

COLOR_BARRA = (0, 200, 0)

COLOR_BARRA_VACIA = (100, 100, 100)


# Parámetros de la resistencia

resistencia_max = 100

resistencia_actual = resistencia_max

consumo_por_uso = 20

regeneracion_por_segundo = 10

tiempo_ultimo_uso = 0
```

```
cooldown = 1000 # milisegundos antes de que empiece a regenerarse
```

```
# Posición y tamaño de la barra
```

```
barra_x = 50
```

```
barra_y = 50
```

```
barra_ancho = 200
```

```
barra_alto = 20
```

```
# Bucle principal
```

```
running = True
```

```
while running:
```

```
    dt = clock.tick(60) # Control de FPS y delta time
```

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

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

```
            running = False
```

```
# Simular acción que consume resistencia al presionar espacio
```

```
keys = pygame.key.get_pressed()
```

```
if keys[pygame.K_SPACE]:
```

```
    if resistencia_actual >= consumo_por_uso:
```

```
        resistencia_actual -= consumo_por_uso
```

```
        tiempo_ultimo_uso = pygame.time.get_ticks()
```

```
# Regenerar resistencia si pasó el cooldown

tiempo_actual = pygame.time.get_ticks()

if tiempo_actual - tiempo_ultimo_uso > cooldown:

    resistencia_actual += regeneracion_por_segundo * (dt / 1000)

    if resistencia_actual > resistencia_max:

        resistencia_actual = resistencia_max


# Dibujar fondo

screen.fill(COLOR_FONDO)


# Dibujar barra vacía (fondo)

pygame.draw.rect(screen, COLOR_BARRA_VACIA, (barra_x, barra_y,
barra_ancho, barra_alto))


# Calcular ancho de la barra actual

ancho_actual = (resistencia_actual / resistencia_max) *
barra_ancho


# Dibujar barra actual

pygame.draw.rect(screen, COLOR_BARRA, (barra_x, barra_y,
ancho_actual, barra_alto))


# Actualizar pantalla

pygame.display.flip()


# Salir
```

```
pygame.quit()
```

```
sys.exit()
```

---

## ¿Qué hace este ejemplo?

- Al presionar **espacio**, se consume resistencia (20 unidades).
- Luego de un pequeño **cooldown de 1 segundo**, la resistencia empieza a regenerarse automáticamente.
- La barra se dibuja en la pantalla y refleja el nivel actual de resistencia.

Día 9:

“Hola, necesito que me ayudes a implementar el clima dinámico dentro del juego”

### Implementación del Clima Dinámico

Hola, para implementar el clima dinámico en tu juego, necesitas seguir estos pasos:

#### 1. Obtener los datos del clima desde la API

Primero, asegúrate de tener la función que obtiene el clima desde la API (parece que ya la tienes en `api.py`):

```
python
```

```
def obtener_clima():
```

```
    try:
```

```
        resp = requests.get(URL_CLIMA, timeout=5)
```

```

    if resp.status_code == 200:

        return resp.json()

except:

    print("No se pudo conectar a la API. Usando clima local...")

    with open(LOCAL_CLIMA, "r") as f:

        return json.load(f)

```

## 2. Crear un módulo simple para manejar el clima

Crea un archivo clima.py:

```
python
```

```
import time
```

```
class Clima:
```

```
    def __init__(self, datos_clima):
```

```
        self.bursts = datos_clima.get("bursts", [])
```

```
        self.indice_actual = 0
```

```
        self.tiempo_inicio = time.time()
```

```
        self.tiempo_burst_actual = 0
```

```
    def obtener_multiplicador(self):
```

```
        """Retorna el multiplicador de velocidad según el clima actual"""
```

```
        tiempo_transcurrido = time.time() - self.tiempo_inicio - self.tiempo_burst_actual
```

```
        if self.indice_actual >= len(self.bursts):
```

```
            return 1.0 # clima normal
```

```
burst_actual = self.bursts[self.indice_actual]
```

```
duracion = burst_actual["duration_sec"]
```

```
# Si pasó la duración del burst actual, pasar al siguiente
```

```
if tiempo_transcurrido >= duracion:
```

```
    self.tiempo_burst_actual += duracion
```

```
    self.indice_actual += 1
```

```
    return self.obtener_multiplicador()
```

```
# Calcular multiplicador según la condición
```

```
condicion = burst_actual["condition"]
```

```
intensidad = burst_actual["intensity"]
```

```
if condicion == "rain":
```

```
    return 1.0 - (0.3 * intensidad) # reduce velocidad hasta 30%
```

```
elif condicion == "storm":
```

```
    return 1.0 - (0.5 * intensidad) # reduce velocidad hasta 50%
```

```
elif condicion == "clouds":
```

```
    return 1.0 - (0.1 * intensidad) # reduce velocidad hasta 10%
```

```
return 1.0
```

```
def obtener_condicion_actual(self):
```

```
    """Retorna la condición climática actual"""
```

```
    if self.indice_actual >= len(self.bursts):
```

```
        return "normal"
```



```
return self.bursts[self.indice_actual]["condition"]
```

### 3. Integrar en el Main.py

En tu archivo principal, después de cargar el mapa:

```
python
```

```
# Cargar clima
```

```
datos_clima = api.obtener_clima()
```

```
clima = Clima(datos_clima.get("data", {}))
```

Luego, en el bucle principal, antes de mover al jugador:

```
python
```

```
# Obtener multiplicador del clima
```

```
m_clima = clima.obtener_multiplicador()
```

Y cuando llames al movimiento del jugador:

```
python
```

```
jugador.mover(dx, dy, tiles, m_clima)
```

### 4. (Opcional) Mostrar el clima en pantalla

Puedes agregar un indicador visual del clima actual en tu bucle de renderizado:

```
python
```

```
# Mostrar clima actual
```

```
condicion = clima.obtener_condicion_actual()
```

```
font_clima = pygame.font.SysFont(None, 28)
```

```
texto_clima = f"Clima: {condicion}"
```

```
screen.blit(font_clima.render(texto_clima, True, (0, 0, 0)), (10, 70))
```

¡Eso es todo! Con esto el clima afectará la velocidad de movimiento del jugador de forma dinámica según los "bursts" que vengan de la API. El sistema automáticamente cambiará entre diferentes condiciones climáticas según las duraciones especificadas.

Día 10: "Hola, estoy haciendo un proyecto en Python con Pycharm, Mi juego ya está algo avanzado y tengo un jugador con podrías decirme cómo puedo cargar y guardar los datos del juego en un archivo, con el nombre del jugador, puntaje y la fecha actual?"

Código actual: ¡Hola! Qué bueno que ya tengas tu juego avanzado 🎮

Si quieres **guardar y cargar los datos del juego** (como el **nombre del jugador**, el **puntaje** y la **fecha actual**) en un archivo, puedes hacerlo fácilmente en **Python** usando archivos de texto o el formato **JSON**, que es muy práctico y legible.

---

## ✅ Recomendación: usar archivos **.json**

### Ventajas:

- Fácil de leer y escribir.
  - Estructurado (como un diccionario).
  - Compatible con muchos lenguajes (por si quieres migrar a futuro).
- 

## 🔧 Ejemplo completo: Guardar y cargar datos del juego en JSON

📁 Archivo: **datos\_juego.json**

### 1. Código para guardar datos:

```
import json

from datetime import datetime

def guardar_datos(nombre_jugador, puntaje):
```

```
datos = {  
    "nombre": nombre_jugador,  
    "puntaje": puntaje,  
    "fecha": datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
}  
  
with open("datos_juego.json", "w") as archivo:  
    json.dump(datos, archivo, indent=4)  
  
print("✅ Datos guardados correctamente.")
```

---

## 2. Código para cargar datos:

```
def cargar_datos():  
    try:  
        with open("datos_juego.json", "r") as archivo:  
            datos = json.load(archivo)  
            return datos  
    except FileNotFoundError:  
        print("⚠️ No se encontró el archivo de guardado.")  
        return None
```

---

## 3. Cómo usarlo en tu juego:

```
# Guardar al final de la partida
guardar_datos("Jugador1", 1200)

# Cargar al iniciar el juego
datos = cargar_datos()

if datos:

    print(f"Último jugador: {datos['nombre']}")

    print(f"Puntaje: {datos['puntaje']}")

    print(f"Fecha: {datos['fecha']}")
```

---

## Resultado esperado del archivo `datos_juego.json`:

```
{

    "nombre": "Jugador1",

    "puntaje": 1200,

    "fecha": "2025-10-04 16:30:12"

}
```

---

## ¿Qué más puedes hacer?

- Guardar **múltiples partidas** (como una lista de jugadores).
- Agregar más campos: nivel, tiempo jugado, vidas, etc.
- Cargar los datos y mostrarlos en un menú o tabla de puntuaciones.