

Prompt 1: Estoy diseñando un sistema de resistencia para mi jugador. La idea es que vaya bajando según el peso que lleve, el tipo de clima y cuánto se mueva, pero también que se recupere si se queda quieto un rato. Cómo podría hacerlo?

```
def consumir_resistencia(self, clima, peso_actual, dt):
    base = 1.5 * dt
    peso = 0.03 * dt if peso_actual > 3 else 0

    efecto_clima = {
        "lluvia": 0.05,
        "viento": 0.03,
        "tormenta": 0.08,
        "calor_extremo": 0.06,
        "soleado": 0.0
    }

    estado_clima = clima.estado.lower() if isinstance(clima.estado, str) else "desconocido"
    clima_factor = efecto_clima.get(estado_clima, 0.0) * dt

    total = base + peso + clima_factor
    self.resistencia = max(0, self.resistencia - total)

def recuperar_resistencia(self, dt):
    recuperacion_rate = 0.5
    if self.resistencia < 100:
        self.resistencia += recuperacion_rate * dt
        self.resistencia = min(100, self.resistencia)
```

Había que juntar todas las situaciones que la afectan y realizar el cálculo.

Prompt 2: Por medio de la API me pasan un json con los pedidos, yo lo que quiero es leer ese json y pasar eso a una lista. Puedo hacer una clase Pedidos y una Pedido? y en la clase Pedidos agregar un metodo procesarPedidos? que me lea el json desde ApiClient

Prompt 3: Para guardar los pedidos, nosotros habíamos visto colas de prioridad, como crees que pueda implementar eso para que mis pedidos se guardan por prioridad

- Nos dio una estructura de cómo podía ser la clase Pedido y Pedidos

```

def procesar_pedidos(self):
    datos = self.api.obtener_trabajos()

    if isinstance(datos, dict) and "data" in datos and isinstance(datos["data"], list):
        self.pedidos = []
        for i, pedido_data in enumerate(datos["data"]):
            pedido = Pedido(
                id=pedido_data["id"],
                payout=pedido_data["payout"],
                priority=pedido_data["priority"],
                pickup=tuple(pedido_data["pickup"]),
                dropoff=tuple(pedido_data["dropoff"]),
                weight=pedido_data["weight"],
                deadline=pedido_data["deadline"],
                release_time=pedido_data["release_time"]
            )
            heapq.heappush(self.pedidos, (-pedido.priority, i, pedido))
        return True
    else:
        print("Error: Los datos de la API no tienen el formato esperado.")
        return False

```

Y agregamos lo del heap para la prioridad de las entregas.

Prompt 4: Yo tengo una lista de pedidos pero como puedo hacer para que mi jugador decida que pedidos quiere hacer y que ese pedido que decida se guarde en una clase inventario con una lista doblemente enlazada para que se pueda recorrer tanto de atrás para adelante, pero que si supera un peso máximo no se pueda guardar el pedido en el inventario.

```

def agregar_pedido(self, pedido):
    if self.peso_actual + pedido.weight > self.peso_maximo:
        return False # No se puede agregar, excede el peso
    nuevo = PedidoNodo(pedido)
    if not self.head:
        self.head = self.tail = self.actual = nuevo
    else:
        self.tail.next = nuevo
        nuevo.prev = self.tail
        self.tail = nuevo
    self.peso_actual += pedido.weight
    return True

```

Utilizamos la estructura de clases más las mejoras sugeridas.

Prompt 5: Cómo hago para cuando salgan los pedidos el jugador los pueda aceptar o rechazar

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False

    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_a:
            pedido_a_aceptar = pedidos.obtener_siguiente_pedido()
            if pedido_a_aceptar:
                if trabajador.inventario.agregar_pedido(pedido_a_aceptar):
                    pedidos.aceptar_pedido()
                    pedidos_tratados += 1
                    incluido = True
                    if inventario_modo == '0':
                        inventario = trabajador.inventario.visualizar_por_entrega()
                    else:
                        inventario = trabajador.inventario.visualizar_por_prioridad()
                else:
                    incluido = False

            elif event.key == pygame.K_r:
                if pedidos.pedidos:
                    pedido_rechazado = pedidos.rechazar_pedido()
                    if pedido_rechazado:
                        trabajador.estado.modificar_reputacion(-3)
                        penalizaciones += 3
                        pedidos_tratados += 1
```

Nos explicó cómo funciona event.key

Prompt 6: Cómo hago para que se pinte una celda en específico

```
def resaltar_celda(self, x, y, color, texto=None):
    font_numero = pygame.font.SysFont("Arial", 12, bold=True)

    x_pixel = x * self.cell_size
    y_pixel = y * self.cell_size

    s = pygame.Surface((self.cell_size, self.cell_size), pygame.SRCALPHA)
    s.fill(color)
    self.screen.blit(s, (x_pixel, y_pixel))

    if texto:
        color_texto = (255, 255, 255)
        superficie_texto = font_numero.render(texto, True, color_texto)

        x_centrado = x_pixel + (self.cell_size - superficie_texto.get_width()) // 2
        y_centrado = y_pixel + (self.cell_size - superficie_texto.get_height()) // 2
        self.screen.blit(superficie_texto, (x_centrado, y_centrado))
```

Prompt 7: Tengo un proyecto que es de un juego, cuando la persona gane o pierda quisiera lanzar una ventana que diga "Felicidades, has ganado" o "Has perdido", como lo puedo hacer para que cuando se cumplan las condiciones abra la ventana. Debería de hacer una clase que se llame estado_de_juego, que determine si gane o perdí y que se cree la ventana ahí?

- ¡Absolutamente! Entiendo que quieres la guía paso a paso para implementar la ventana de victoria/derrota en tu código Pygame usando la solución de tkinter.messagebox.

```
def mostrar_estado_final(resultado):  
    root = tk.Tk()  
    root.withdraw()  
  
    if resultado == "victoria":  
        titulo = "VICTORIA"  
        mensaje = f"¡Felicidades! Has alcanzado la meta"  
        messagebox.showinfo(titulo, mensaje)  
    elif resultado == "derrota":  
        titulo = "DERROTA"  
        mensaje = f"Perdiste, no alcanzaste los objetivos"  
        messagebox.showinfo(titulo, mensaje)  
  
    root.destroy()
```

Luego nada más se llama la función dentro del condicional y se imprime la ventana.

Prompt 8: Antes de que termine de entregar los pedidos ya me tira la derrota.

```
if trabajador.estado.ingresos >= params["goal"]:  
    running = False  
    mostrar_estado_final("victoria")  
elif trabajador.estado.reputacion < 20:  
    running = False  
    mostrar_estado_final("derrota")  
elif (total_pedidos == pedidos_tratados  
      and trabajador.estado.ingresos < params["goal"]  
      and trabajador.inventario.esta_vacia()):  
    running = False  
    mostrar_estado_final("derrota")
```

Había que verificar que el inventario estuviera vacío, para que no tirara el mensaje de que ya había perdido antes de que terminara los pedidos.

Prompt 9: Me apareció este error al debugear: “can't subtract offset-naive and offset-aware datetimes” qué significa y cómo se soluciona?

Ese error ocurre cuando intentás restar dos objetos `datetime` en Python, pero uno tiene zona horaria (offset-aware) y el otro no (offset-naive). Python no permite esa operación porque no puede garantizar que la diferencia sea precisa. Para solucionarlo, tenés que asegurarte de que ambos objetos tengan zona horaria, por ejemplo usando `datetime.fromtimestamp(..., tz=timezone.utc)` para que ambos estén en UTC.

Prompt 10: tengo pedidos que tienen un tiempo límite para ser entregados, como tengo el tiempo de liberación y el deadline, quiero calcular cuánto tiempo le queda al jugador para entregar el pedido, cómo puedo hacerlo?

Podés agregar un método dentro de tu clase `Pedido` que reciba el tiempo actual del juego y devuelva cuántos segundos faltan para el deadline. Por ejemplo:

Python

 Copiar

```
def tiempo_restante(self, tiempo_juego):  
    restante = self.deadline - tiempo_juego  
    return max(0, restante)
```

Esto te permite mostrar visualmente el tiempo restante y aplicar penalizaciones si el valor llega a cero.

Prompt 11: con pygame cómo hago que el jugador pueda aceptar o rechazar pedidos?

Python

 Copiar

```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_a: # Aceptar

            pedido = pedidos.obtener_siguiente_pedido()
            if trabajador.inventario.agregar_pedido(pedido):
                pedidos.aceptar_pedido()
        elif event.key == pygame.K_r: # Rechazar

            pedido = pedidos.rechazar_pedido()
            trabajador.estado.modificar_reputacion(-3)
```

Esto te permite integrar decisiones del jugador directamente en el flujo del juego.

Prompt 12: Me apareció este error: could not convert string to float: '2025-09-01T12:10Z' qué significa?

Ese error ocurre cuando el código intenta convertir una cadena de texto que representa una fecha (como "2025-09-01T12:10Z") directamente a float, lo cual no tiene sentido semántico. Esto suele pasar en bloques except mal diseñados, donde se asume que cualquier valor puede ser numérico. Para evitarlo, hay que validar el tipo de dato antes de convertir, y asegurarse de que las fechas se procesen con datetime y no con float.

Prompt 13: Estoy usando fechas como "2025-09-01T12:10Z" que vienen en formato ISO, pero el sistema me lanza "Formato de fecha no reconocido". ¿Por qué pasa eso si el formato es estándar?

Aunque el formato ISO es estándar, la "Z" al final indica zona horaria UTC, y funciones como `datetime.fromisoformat()` no la aceptan directamente. Para que funcione, hay que reemplazar "Z" por "+00:00" antes de convertir. Por ejemplo:

Python

 Copiar

```
fecha = fecha.replace("Z", "+00:00")
fecha_dt = datetime.fromisoformat(fecha)
```

Esto asegura compatibilidad con el parser de Python y evita el error.






Prompt 14: Cómo hago para sacar la posición exacta de mi jugador?

```
def guardar_partida(self, trabajador, clima, pedidos):
    center_x = trabajador.trabajadorRect.centerx
    center_y = trabajador.trabajadorRect.centery

    columna = center_x // trabajador.cell_size
    fila = center_y // trabajador.cell_size
    estado_juego = {
        'pos_x': fila,
        'pos_y': columna,
        'center_x': center_x,
        'center_y': center_y,
        'cell_size': trabajador.cell_size,
```


Nos sugirieron sacar la casilla exacta donde se encuentra el jugador de esa manera e igualmente guardar las coordenadas que usa pygame y el tamaño de la celda.

Prompt 15:

Cancelación de pedidos		Se puede rechazar pedidos, pero no hay penalización por cancelar pedidos ya aceptados.
Visualización de entregas puntuales/tardías		Se imprime en consola, pero no se muestra visualmente en el panel lateral.
Puntajes finales		puntajes.py y puntaje.py existen, pero el archivo puntajes.json no tiene formato válido. Falta cálculo y visualización del puntaje.
Orden de entrega manual		Se puede navegar con siguiente y anterior, pero no hay función para reordenar pedidos.
Guardado/carga de partida		No se ha implementado aún. Requiere persistencia en /saves/slot1.sav.

Prompt 16:

has una lista de TODOS los requerimientos del proyecto, dando cuáles ya se cumplen, cuáles se deben modificar y cuáles aún no están implementados

 Copilot ⓘ


¡Perfecto! Ya revisé todos los archivos que has compartido y el documento del proyecto **Courier Quest**. Aquí tienes una lista completa de los requerimientos, dividida en tres categorías:

✅ **REQUERIMIENTOS CUMPLIDOS**

Requisito	Estado	Detalles
Ciudad como cuadrícula	✅	<code>mapa.py</code> procesa el mapa en forma de matriz con calles, edificios y parques.
Movimiento por casillas	✅	El personaje se mueve celda por celda

Prompt 17:





cómo puedo realizar una impresión de una ventana emergente con información para que el usuario seleccione una opción entre las opciones impresas

 Copilot ⓘ






Para mostrar una **ventana emergente con opciones** para que el usuario seleccione una, puedes usar **tkinter**, que ya estás utilizando en tu proyecto. Aquí te muestro cómo hacerlo con un **diálogo personalizado** usando `tkinter.simpledialog` o una **ventana con botones** para múltiples opciones.

✅ **Opción 1: Usar `tkinter.simpledialog.askstring` o `askinteger`**

Esto sirve si quieres que el usuario escriba su elección:

    Python

Prompt 18:

Cancelación de pedidos		Se puede rechazar pedidos, pero no hay penalización por cancelar pedidos ya aceptados.
Visualización de entregas puntuales/tardías		Se imprime en consola, pero no se muestra visualmente en el panel lateral.
Puntajes finales		puntajes.py y puntaje.py existen, pero el archivo puntajes.json no tiene formato válido. Falta cálculo y visualización del puntaje.
Orden de entrega manual		Se puede navegar con siguiente y anterior, pero no hay función para reordenar pedidos.
Guardado/carga de partida		No se ha implementado aún. Requiere persistencia en /saves/slot1.sav.

Prompt 19:

has una lista de TODOS los requerimientos del proyecto, dando cuáles ya se cumplen, cuáles se deben modificar y cuáles aún no están implimentados

 Copilot ⓘ

¡Perfecto! Ya revisé todos los archivos que has compartido y el documento del proyecto Courier Quest. Aquí tienes una lista completa de los requerimientos, dividida en tres categorías:

✓ REQUERIMIENTOS CUMPLIDOS

Requisito	Estado	Detalles
Ciudad como cuadrícula	✓	mapa.py procesa el mapa en forma de matriz con calles, edificios y parques.
Movimiento por casillas	✓	El personaje se mueve celda por celda

Prompt 20:

cómo puedo realizar una impresión de una ventana emergente con información para que el usuario seleccione una opción entre las opciones impresas

Copilot ⓘ

Para mostrar una **ventana emergente con opciones** para que el usuario seleccione una, puedes usar `tkinter`, que ya estás utilizando en tu proyecto. Aquí te muestro cómo hacerlo con un **diálogo personalizado** usando `tkinter.simpledialog` o una **ventana con botones** para múltiples opciones.

✓ Opción 1: Usar `tkinter.simpledialog.askstring` o `askinteger`

Esto sirve si quieres que el usuario escriba su elección:

Python

Prompt 21:

✗ REQUISITOS NO IMPLEMENTADOS

Requisito	Estado	Qué falta
Tabla de puntajes ordenada	✗	<code>puntajes.json</code> no está en formato JSON válido ni ordenado por ingresos.
Archivo README.md	✗	No se ha incluido ni generado.
Registro de prompts y modificaciones	✗	No hay bitácora de uso de IA ni cambios realizados.
Puntos de descanso en el mapa	✗	No hay celdas especiales para recuperación acelerada.
Interpolación de efectos climáticos	✗	No se realiza transición suave entre climas.
Guardado de estado en JSON	✗	No hay función para guardar/cargar estado del juego

Prompt 22:

Hoy

cómo puedo, desde python usando pygame, hacer un if que se ejecute si no se presionó ninguna tecla

Copilot ⓘ

En Pygame, para ejecutar un bloque de código solo cuando no se presiona ninguna tecla, puedes usar el estado del teclado en cada fotograma. Aquí te muestro cómo hacerlo:

Ejemplo básico:

```
15         corriendo = False
16
17     # Obtener el estado de todas las teclas
18     teclas = pygame.key.get_pressed()
19
20     # Verificar si ninguna tecla está presionada
21     if not any(teclas):
```

Prompt 23:

		usando pulsaciones de teclas.
Clima dinámico	✓	datos_clima.py implementa cambios climáticos cada 45-60 segundos con cadena de Markov.
Inventario con peso máximo	✓	inventario.py controla el peso y rechaza pedidos que exceden el límite.
Reputación	✓	estado_trabajador.py gestiona reputación y aplica cambios por entregas.
Reputación afecta pagos	✓	pedido.py aplica bonus del 5% si la reputación ≥ 90.
Reputación crítica	✓	El juego termina si la reputación < 20.
Pedidos con prioridad	✓	pedidos.py usa heapq para priorizar pedidos por priority.
Recibe y entrega por	✓	pedido.py verifica si el trabajador está

Prompt 24:

🔧 REQUISITOS PARCIALMENTE IMPLEMENTADOS / A MEJORAR		
Requisito	Estado	Qué falta o mejorar
Recuperación de resistencia	🔧	Se recupera, pero no se implementan puntos de descanso (+10/seg). Actualmente es +0.5/seg máximo.
Clima afecta resistencia	🔧	estado_trabajador.py tiene factores, pero no usa clima.efecto_trabajador() ["resistencia"] directamente.
Transición suave entre climas	🔧	datos_clima.py cambia el clima abruptamente. Falta interpolación gradual (3-5 segundos).
Cancelación de pedidos	🔧	Se puede rechazar pedidos, pero no hay penalización por cancelar pedidos va

Prompt 25:

```
20  # Verificar si ninguna tecla está presionada
21  if not any(teclas):
22      print("Ninguna tecla está siendo presionada")
23
24  # Actualizar la pantalla
25  pantalla.fill((0, 0, 0))
26  pygame.display.flip()
27
28  # Salir de Pygame
29  pygame.quit()
```

Mostrar más líneas

Explicación:

- `pygame.key.get_pressed()` devuelve una lista de booleanos, donde cada índice representa una tecla.
- `any(teclas)` devuelve True si alguna tecla está presionada.
- `not any(teclas)` significa que ninguna tecla está presionada.

Prompt 26:

Esto sirve si quieres que el usuario escriba su elección:

```
1 import tkinter as tk
2 from tkinter import simpledialog
3
4 def seleccionar_opcion():
5     root = tk.Tk()
6     root.withdraw() # Oculta la ventana principal
7
8     opcion = simpledialog.askstring("Seleccionar opción", "Escribe una opción: A, B o C")
9
10    root.destroy()
11    return opcion
12
```

✓ Opción 2: Crear una ventana con botones para cada opción

Esto es más visual y amigable:

Prompt 27:

en la parte de persistencia de archivos a que crees que se refieren con archivo binario

En la parte de **persistencia de archivos**, cuando el documento dice:

• Guardado binario: /saves/slot1.sav"

se refiere a que el **estado del juego** (por ejemplo: posición del jugador, clima actual, pedidos activos, resistencia, reputación, etc.) debe poder **guardarse en un archivo binario** —es decir, un archivo que no almacena texto legible, sino los datos en formato **binario** (bytes).

Prompt 28: Debería de guardar todas las partidas o solo la anterior?

Buena pregunta 🤖

En el documento del proyecto dice específicamente:

• Guardado binario: `/saves/slot1.sav`

Eso sugiere que solo se requiere un único "slot" de guardado, es decir, una partida activa a la vez (como un autosave o última partida guardada).

- Cada vez que el jugador guarde, ese archivo (`slot1.sav`) se sobrescribe con el nuevo estado.
- Y cuando carga partida, simplemente se lee ese mismo archivo.