

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.