

IA usada	Prompt
Copilot	<p>Estoy programando en Python una aplicación. Sin embargo, nunca he usado una API en un lenguaje de programación. Me surge la duda: si yo cargo un API en Python que contiene archivos de formato JSON, ¿debo realizar una clase para procesarlos y así trabajar con los archivos que se obtengan? Considere que cada archivo JSON tiene información distinta y, por lo tanto, no se pueden guardar en una sola clase, ya que cada uno tiene un propósito distinto.</p> <p>Si la estructura de los archivos del API pueden cambiar de forma dinámica (es decir, se pueden modificar en el tiempo y en sus datos), ¿de qué forma puedo trabajarlos? ¿Existe una librería para importarlos de forma dinámica? Si es el caso, ¿cuál es la librería o forma que me permite trabajarlo de forma eficiente? ¿Es necesario crear una clase para ello?</p> <p>¿Cuáles son las mejores prácticas para trabajar con APIs?</p> <p>Conclusión: Usar pydantic para manejar los archivos JSON</p>
Copilot	<p>Estoy programando con Python en Visual Studio Code. Quiero usar una API de manera dinámica; es decir, que puede cambiar el contenido de los archivos que debo importar. Los archivos que posee esta API son archivos de tipo JSON. Mi programa debe ser capaz de importar estos archivos de manera local a la computadora. Además, estos deben trabajar tanto de manera online conectándose con la API, y de manera offline por medio de la caché con los últimos archivos cargados. ¿Qué me permitiría realizar esto en Python? En este caso, ¿se usa la librería pydantic?</p> <p>¿Cómo puedo aplicarlo con la librería import junto con la librería pydantic? El link que quiero utilizar es: <a href="https://tigerds-api.kindflower-ccaf48b6.eastus.azurecontainerapps.io">https://tigerds-api.kindflower-ccaf48b6.eastus.azurecontainerapps.io</a></p> <p>¿Y cómo yo sé qué tipo de parámetros maneja? Esto por medio de la lectura del archivo JSON?</p>
Copilot	Estoy trabajando con diferentes archivos de Python en Visual Studio Code. ¿Cómo puedo importar el código de un archivo .py a otro archivo .py?
	Por qué debo utilizar el __init__.py al crear una carpeta con diferentes variables? Para qué sirve tenerla en Python?

	<p>¿Qué ocurre con este error? from .Pedido import Pedido  ImportError: attempted relative import with no known parent package. Esto ocurre en la siguiente clase llamada __init__.py. Esto está construido de la siguiente manera: from . import Pedido from . import CityMap from .Clima import ResultadoClima __all__ = ["Pedido", "CityMap", "ResultadoClima"]</p> <p>Existe un __init__.py en la carpeta Imports. Es uno aparte del __init__.py que está dentro de Clima, la carpeta que está dentro de Imports. ¿Puede que esto esté dando problemas con el pathing? La estoy construyendo de la siguiente manera: from Imports.Pedido import Pedido from Imports.CityMap import CityMap from Imports.Clima import ResultadoClima __all__ = ["Pedido", "CityMap", "ResultadoClima"]</p>
Copilot	<p>Estoy trabajando en un proyecto de Python. El programa consiste en construir un juego usando una librería de desarrollo de juegos, usando la librería Pygame. Simula a un repartidor que debe aceptar y completar pedidos en una ciudad, gestionando tiempos de entrega, clima, inventario y prioridades. Yo ya hice un folder que funciona como paquete donde se tiene el API. También tengo un folder models, donde tengo las clases que corresponden al API. Ahora, quiero hacer la clase Repartidor. Qué factores debo considerar? Considere que el API trae solamente la información de clima, ciudad y pedidos. Todo lo demás lo debo construir yo.</p>
ChatGPT	<p>I am learning to use Python. Specifically, learn to use Pygame. I already have a structure of the API and other elements, but I first want to learn about the sprites, controls, screen, buttons and other relevant elements I should know before I get started. What do you recommend to explore more about this library?</p> <p>How do I make the character move with wasd? How do I make the screen bigger, and which size is more appropriate for general use?</p> <p>For now, keep it in one direction. Now, I want to learn how to make a starting screen, before the cat moves. How can I do it?</p>
ChatGPT	<p>Estoy haciendo un proyecto utilizando Pygame en el lenguaje de programación Python. Estoy utilizando como IDE Visual Studio Code. Ya hice los sprites para el mapa, que se carga con un API. Sin embargo, tengo una duda. Si tengo un personaje que quiero que se mueva por el mapa, cómo debería manejar los layers o capas en PyGame, de manera tal que, al momento de estar en un lugar del mapa, cambie el comportamiento del personaje?</p> <p>Antes de eso, quisiera que observes esto en específico: Clima: cambia automáticamente cada 45-60 segundos siguiendo una</p>

	<p>cadena de Markov. o Lluvia, tormentas, viento o calor extremo reducen la velocidad y aumentan el consumo de resistencia. o La transición entre climas es progresiva, para que los cambios se sientan naturales. Yo tengo en el proyecto una carpeta llamada models. En ella, estoy guardando una parte de climas, cargada del API. En ella, se divide en tres partes: una que es ClimaData.py, otra que es CondicionClima.py, y la última, que es ResultadoClima.py. Eso debe conectar, de alguna manera, al mapa del juego, conozco que debo hacerlo con una cadena de Markov, pero no sé cómo funciona.</p> <p>Esa clase WeatherSystem en cuál carpeta debe ir? Yo tengo estas carpetas hasta el momento: api (con un ManejadorAPI que carga los datos del API), cache (que guarda los datos que vienen del API de manera offline), models (que ahí se cargan en clases los datos del API). ¿Debería ir en alguna de ellas o en una nueva?</p> <p>¿Hay alguna forma de hacer los sprites de score desde el mismo programa o no?</p> <p>Me está dando un error: ModuleNotFoundError</p>
ChatGPT	<p>Estoy haciendo este código en Python usando Visual Studio Code (adjunto stats_module.py). En él, el estado_clima no es leído. Revisé los nombres correctos (__init__) junto con las direcciones. Quiero que me ayudes a encontrar el error para que lo lea de forma apropiada. Por favor, revisa bien el código y dame alguna forma de probarlo.</p> <p>*Se adjunta el código de la resistencia*</p> <p>Realicé este código, que se encarga de cambiar la resistencia, usando Visual Studio code, con el lenguaje de programación Python. Este debe realizar lo siguiente: Resistencia: se representa con una barra de 0–100. o Baja con el movimiento, sobre todo si lleva mucho peso o si el clima es adverso. o Si llega a 0, el jugador queda exhausto y no puede moverse hasta recuperarse al 30%. Esto se hace usando la herramienta Pygame. Ahora, lo que pasa es que quiero probarlo en el main, de manera tal que no me de errores. ¿Puedes ayudarme?</p> <p>El código que me pasaste no se adapta a lo que estoy escribiendo en el resto del programa. Revisa los métodos utilizados, ya que debo estar seguro que todos funcionan como corresponden.</p> <p>Me explicas qué es una cadena de markov? Cómo funciona en programación? De paso, dame algunos ejemplos de</p>

	<p>implementación de cadenas de markov.</p> <p>Quisiera implementarlo de esta forma: Clima: cambia automáticamente cada 45-60 segundos siguiendo una cadena de Markov. o Lluvia, tormentas, viento o calor extremo reducen la velocidad y aumentan el consumo de resistencia. o La transición entre climas es progresiva, para que los cambios se sientan naturales.</p> <p>clouds: ×0.98 - rain_light: ×0.90 - rain: ×0.85 - storm: ×0.75 - fog: ×0.88 - wind: ×0.92 - heat: ×0.90 - cold: ×0.92</p> <p>Si el clima actual es nublado, hay un 30% de que vuelva a despejado, 50% de que siga nublado y 20% de que cambie a lluvia. Además de la condición, cada ráfaga tiene un nivel de intensidad (0–1). Para que el cambio entre climas no sea brusco, la transición entre la condición actual y la nueva debe hacerse en 3–5 segundos interpolando los efectos (ej. velocidad del jugador, visibilidad).</p>
ChatGPT	<p>Estoy haciendo un proyecto utilizando Pygame en el lenguaje de programación Python. Estoy utilizando como IDE Visual Studio Code. Ya hice los sprites para el mapa, que se carga con un API. Sin embargo, tengo una duda. Si tengo un personaje que quiero que se mueva por el mapa, cómo debería manejar los layers o capas en PyGame, de manera tal que, al momento de estar en un lugar del mapa, cambie el comportamiento del personaje?</p> <p><i>*Se adjunta enunciado*</i></p> <p>Quiero que analices el enunciado. Para manejar toda la lógica de PyGame, quiero hacerlo en una carpeta aparte del main.py. ¿Es posible hacerlo?</p> <p>Espera. Antes de eso, quisiera que observes esto en específico: Clima: cambia automáticamente cada 45-60 segundos siguiendo una cadena de Markov. o Lluvia, tormentas, viento o calor extremo reducen la velocidad y aumentan el consumo de resistencia. o La transición entre climas es progresiva, para que los cambios se sientan naturales. Yo tengo en el proyecto una carpeta llamada models. En ella, estoy guardando una parte de climas, cargada del API. En ella, se divide en tres partes: una que es ClimaData.py, otra que es CondicionClima.py, y la última, que es ResultadoClima.py. Eso debe conectar, de alguna manera, al mapa del juego, conozco que debo hacerlo con una cadena de Markov, pero no sé cómo funciona. Note que esto es el formato JSON que carga del API: { "version": "1.0", "data": { "city": "TigerCity", "initial": { "condition": "clear", "intensity": 0 }, "conditions": [ "clear",</p>

```
"clouds", "rain_light", "rain", "storm", "fog", "wind", "heat", "cold"
], "transition": { "clear": { "clear": 0.6, "clouds": 0.3, "rain": 0.1 },
"clouds": { "clear": 0.3, "clouds": 0.5, "rain": 0.2 }, "rain": {
"clouds": 0.4, "rain": 0.4, "storm": 0.2 }, "rain_light": { "clouds":
0.4, "rain_light": 0.4, "rain": 0.2 }, "storm": { "rain": 0.5, "clouds":
0.3, "storm": 0.2 }, "fog": { "fog": 0.5, "clouds": 0.3, "clear": 0.2 },
"wind": { "wind": 0.5, "clouds": 0.3, "clear": 0.2 }, "heat": { "heat":
0.5, "clear": 0.3, "clouds": 0.2 }, "cold": { "cold": 0.5, "clear": 0.3,
"clouds": 0.2 } } } },
```

Mira cómo están construidas esas tres clases y dime si aplica lo que me dices o no: from pydantic import BaseModel class

```
CondicionClima(BaseModel): condition: str intensity: intfrom
typing import List, Dict from pydantic import BaseModel from
.CondicionClima import CondicionClima class
ClimaData(BaseModel): city: str initial: CondicionClima
conditions: List[str] transition: Dict[str, Dict[str, float]]from
pydantic import BaseModel from .ClimaData import ClimaData
class ResultadoClima(BaseModel): version: str data: ClimaData
```

Resultado:

```
import random, time
from .ResultadoClima import ResultadoClima
```

class WeatherSystem:

```
def __init__(self, resultado_clima: ResultadoClima):
```

```
    self.data = resultado_clima.data
```

```
    self.current = self.data.initial.condition
```

```
    self.intensity = self.data.initial.intensity
```

```
    self.last_change = time.time()
```

```
    self.duration = random.randint(45, 60) # duración en
```

segundos

```
    self.transition_time = 5 # segundos de transición suave
```

```
    self.old_multiplier = 1.0
```

```
    self.new_multiplier = 1.0
```

```
    self.transition_start = None
```

```
def next_condition(self):
```

```
    probs = self.data.transition[self.current]
```

```
    choices = list(probs.keys())
```

```
    weights = list(probs.values())
```

```
    self.current = random.choices(choices, weights=weights)[0]
```

```
    # aquí podrías ajustar la intensidad también
```

```
    return self.current
```

```
def update(self):
```

```
    now = time.time()
```

```
    # Si ya pasaron 45–60s, cambiar de condición
```

```
    if now - self.last_change > self.duration:
```

```
self.last_change = now
self.duration = random.randint(45, 60)
self.old_multiplier = self.get_multiplier(self.current)
self.current = self.next_condition()
self.new_multiplier = self.get_multiplier(self.current)
self.transition_start = now
```

```
def get_multiplier(self, condition):
```

```
    multipliers = {
        "clear": 1.00,
        "clouds": 0.98,
        "rain_light": 0.90,
        "rain": 0.85,
        "storm": 0.75,
        "fog": 0.88,
        "wind": 0.92,
        "heat": 0.90,
        "cold": 0.92
    }
    return multipliers.get(condition, 1.0)
```

```
def get_current_multiplier(self):
```

```
    """Devuelve el multiplicador actual interpolado."""
    if self.transition_start:
        elapsed = time.time() - self.transition_start
        t = min(1.0, elapsed / self.transition_time)
        return self.old_multiplier + (self.new_multiplier -
self.old_multiplier) * t
    return self.new_multiplier
```

Esa clase WeatherSystem en cuál carpeta debe ir? Yo tengo estas carpetas hasta el momento: api (con un ManejadorAPI que carga los datos del API), cache (que guarda los datos que vienen del API de manera offline), models (que ahí se cargan en clases los datos del API). ¿Debería ir en alguna de ellas o en una nueva?

Mira que ya hay datos dados, por lo que puede tomar provecho de eso. Considere todo lo que hemos hablado hasta ahora. Quiero que me des todas las clases y elementos que deben ir en esa carpeta game. Considere que tengo otra carpeta que se llama "sprites", en la cual se cargan los sprites del mapa. Además, yo sé que debo acomodar eso en la lógica, entonces ayudame a hacerlo. Esto estaba incluido de forma provisional en el main.py. Acomodalo, según veas necesario: import pygame from api.ManejadorAPI import ManejadorAPI import json from pathlib import Path from models.CityMap import CityMap def main(): # --- Configuración --  
- TILE\_WIDTH = 40 TILE\_HEIGHT = 44 # Colores para cualquier tile que no tenga sprite asignado TILE\_COLORS = {  
"default": (150, 150, 150) # Gris } # --- Rutas base --- BASE\_DIR

```

= Path(_file_).resolve().parent # Carpeta donde está este main.py
CACHE_DIR = BASE_DIR / "cache" SPRITES_DIR =
BASE_DIR / "sprites" # --- Cargar datos del mapa --- with
open(CACHE_DIR / "map.json", "r", encoding="utf-8") as f: data
= json.load(f) city_map = CityMap(**data) # --- Inicializar Pygame
--- pygame.init() screen = pygame.display.set_mode(
(city_map.width * TILE_WIDTH, city_map.height *
TILE_HEIGHT) )
pygame.display.set_caption(city_map.city_name) clock =
pygame.time.Clock() # --- Función para cargar y escalar sprites ---
def cargar_sprite(nombre_archivo): ruta = SPRITES_DIR /
nombre_archivo imagen =
pygame.image.load(ruta).convert_alpha() return
pygame.transform.scale(imagen, (TILE_WIDTH,
TILE_HEIGHT)) # --- Cargar sprites --- sprite_B =
cargar_sprite("Spr_edificio1.png") sprite_C =
cargar_sprite("Spr_acera.png") sprite_P =
cargar_sprite("Spr_parque.png") # --- Bucle principal --- running =
True while running: for event in pygame.event.get(): if event.type
== pygame.QUIT: running = False screen.fill((0, 0, 0)) for y, fila in
enumerate(city_map.tiles): for x, code in enumerate(fila): rect =
pygame.Rect( x * TILE_WIDTH, y * TILE_HEIGHT,
TILE_WIDTH, TILE_HEIGHT ) if code == "B":
screen.blit(sprite_B, rect.topleft) elif code == "C":
screen.blit(sprite_C, rect.topleft) elif code == "P":
screen.blit(sprite_P, rect.topleft) else: color =
TILE_COLORS.get(code, TILE_COLORS["default"])
pygame.draw.rect(screen, color, rect) pygame.draw.rect(screen,
(50, 50, 50), rect, 1) # Borde pygame.display.flip() clock.tick(60)
pygame.quit() if _name_ == "_main_": main()

```

Ok. En base en esas clases, cuál es la más fácil de construir? Es decir, cuál es la que menos debería afectarme en la lógica que tengo hasta el momento, para ir probando si tiene un buen funcionamiento o no. Quiero que me construyas esas clase en específico primero. Después, vemos las otras.

Haz ahora map\_renderer.py. Sin embargo, quiero probar que todo está bien. Entonces, cambia la lógica del main, para probarlo de una vez, cuando lo implemente. Entonces, quiero que hagas map\_renderer.py. Note que la estructura que te di antes del proyecto está un poco modificada. Lo que cambió es lo siguiente: hice una carpeta llamada "src", la cual tiene como subcarpetas la carpeta "api", "game" y "models". Dentro de ella NO está la carpeta de sprites. Eso está en una carpeta aparte llamada "sprites", en el mismo nivel que src en la jerarquía de las carpetas.

Resultado:

```

# main.py
import pygame
import json
from pathlib import Path

from src.models.CityMap import CityMap
from src.game.map_renderer import MapRenderer

def main():
    # --- Configuración ---
    TILE_WIDTH = 40
    TILE_HEIGHT = 44

    # --- Rutas base ---
    BASE_DIR = Path(__file__).resolve().parent
    CACHE_DIR = BASE_DIR / "cache"
    SPRITES_DIR = BASE_DIR / "sprites" # ojo: está al mismo
    nivel que src

    # --- Cargar datos del mapa ---
    with open(CACHE_DIR / "map.json", "r", encoding="utf-8") as
f:
        data = json.load(f)
        city_map = CityMap(**data)

    # --- Inicializar Pygame ---
    pygame.init()
    screen = pygame.display.set_mode(
        (city_map.width * TILE_WIDTH, city_map.height *
TILE_HEIGHT)
    )
    pygame.display.set_caption(city_map.city_name)
    clock = pygame.time.Clock()

    # --- Crear el renderer ---
    renderer = MapRenderer(city_map, SPRITES_DIR,
TILE_WIDTH, TILE_HEIGHT)

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

        screen.fill((0, 0, 0)) # limpiar pantalla
        renderer.draw(screen) # dibujar mapa
        pygame.display.flip()

```



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

```
pygame.quit()
```

```
if __name__ == "__main__":  
    main()
```

No quiero que hagas player. Me está dando un error: from src.models.CityMap import CityMap. ModuleNotFoundError: No module named 'src'

el \_\_init\_\_.py debe ir tanto en src como en sus subcarpetas?

Habia un error de escritura. Sin embargo, ahora, el problema de map\_renderer es que da el siguiente error: ModuleNotFoundError: no module named 'src'

Ahora, no sale ese error. Sin embargo, no se carga la pantalla de pygame.

Ya logré arreglar el error! Ahora, quisiera seguir con el proyecto. Ahora, necesito hacer la próxima clase más sencilla, después de map\_renderer.py y score.py. Recuerdo que me habías sugerido preparar los hooks en job\_manager.py a partir de lo hecho en score.py. No sé si empezamos con eso o debo preparar otra clase antes, que sea más sencilla.

Me da [stats.py](#):

Ya logré arreglar el error! Ahora, quisiera seguir con el proyecto. Ahora, necesito hacer la próxima clase más sencilla, después de map\_renderer.py y score.py. Recuerdo que me habías sugerido preparar los hooks en job\_manager.py a partir de lo hecho en score.py. No sé si empezamos con eso o debo preparar otra clase antes, que sea más sencilla.

Me funciona hasta el momento. Hice el stats.py como me dijiste. Lee con precisión y todo lo que se deba implementar y revisa de nuevo si stats.py que me pasaste cumple con el enunciado del archivo PDF. Hecho eso, escribe reputation.py. Antes de implementar el reputation.py, quiero probar en main lo puesto en stats.py. Dame el código para hacerlo. Asegurate una vez más que coincida con el enunciado del pdf y cumple con lo mencionado previamente.

Aquí me da un error al probarlo: consumo = stats.consume\_for\_move(cells=5, weight\_total=2,

<div data-bbox="512 208 1375 282" data-label="Text"> <p>climate_condition="clear"). En él, da el siguiente error: got an unexpected keyword argument 'cells'</p> </div> <div data-bbox="512 318 1375 795" data-label="Code"> <pre> Resultado: def consume_for_move(self, cells: int = 1, weight_total: float = 0.0, climate_condition: str = "clear") -&gt; float:     """     Consume resistencia por moverse 'cells' celdas. Devuelve la     cantidad consumida.     El caller (p. ej. Player.update) debe usar esto cuando se mueve.     """     per_cell = self.consumption_per_cell(weight_total, climate_condition)     amount = per_cell * max(0, int(cells))     self._do_consume(amount)     return amount </pre> </div> <div data-bbox="512 866 1375 1637" data-label="Text"> <p>Me da el siguiente error: === PRUEBA DEL SISTEMA DE RESISTENCIA === Inicial: Resistencia=100.0, Estado=normal, Factor=1.0, Puede mover=True Traceback (most recent call last): File "c:\Users\Andrey\Documents\Programas Estructuras de datos\Proyecto-Datos\main.py", line 97, in &lt;module&gt; test_stats() File "c:\Users\Andrey\Documents\Programas Estructuras de datos\Proyecto-Datos\main.py", line 18, in test_stats consumo = stats.consume_por_mover(celdas=5, peso_total=2, condicion_clima="clear") ~~~~~ File "c:\Users\Andrey\Documents\Programas Estructuras de datos\Proyecto-Datos\src\game\statistics.py", line 41, in consume_por_mover por_celda = self.consumo_por_celda_total(peso_total, condicion_clima) ~~~~~ File "c:\Users\Andrey\Documents\Programas Estructuras de datos\Proyecto-Datos\src\game\statistics.py", line 37, in consumo_por_celda_total clima_extra = self.extras_clima.get(condicion_clima, 0.0) ~~~~~ AttributeError: 'NoneType' object has no attribute 'get'</p> </div> <div data-bbox="512 1673 941 1966" data-label="Code"> <pre> Me da este archivo <a href="#">stats.py</a>: from dataclasses import dataclass from typing import Literal  @dataclass class Stats:     resistencia_max: float = 100.0     resistencia: float = 100.0 </pre> </div>
--

```
recuperacion_threshold: float = 30.0
consumo_por_celda: float = 0.5
peso_extra_por_unidad: float = 0.2
extras_clima: dict = None
recuperacion_rate_idle: float = 5.0
recuperacion_rate_rest_point: float = 10.0
```

```
_exhaust_lock: bool = False
```

```
def __post_init__(self):
    # 🙌 Aquí se asegura que extras_clima no quede en None
    if self.extras_clima is None:
        self.extras_clima = {
            "rain": 0.1,
            "wind": 0.1,
            "rain_light": 0.1,
            "storm": 0.3,
            "heat": 0.2,
            "clear": 0.0,
            "clouds": 0.0,
            "fog": 0.0,
            "cold": 0.0,
        }
```

Al agregar esto en test\_stats(), da como resultado None:  
print("Extras\_clima inicial:", stats.extras\_clima)

Resultado: from dataclasses import dataclass  
from typing import Literal

```
@dataclass
class Stats:
    resistencia_max: float = 100.0
    resistencia: float = 100.0

    recuperacion_threshold: float = 30.0
    consumo_por_celda: float = 0.5
    peso_extra_por_unidad: float = 0.2
    extras_clima: dict = None
    recuperacion_rate_idle: float = 5.0
    recuperacion_rate_rest_point: float = 10.0
```

```
_exhaust_lock: bool = False
```

```
def __post_init__(self): # 🙌 OJO: este nombre exacto
    if self.extras_clima is None:
        self.extras_clima = {
            "rain": 0.1,
            "wind": 0.1,
```

```
        "rain_light": 0.1,  
        "storm": 0.3,  
        "heat": 0.2,  
        "clear": 0.0,  
        "clouds": 0.0,  
        "fog": 0.0,  
        "cold": 0.0,  
    }
```

```
    def consumo_por_celda_total(self, peso_total: float,  
condicion_clima: str) -> float:  
        base = self.consumo_por_celda  
        peso_extra = 0.0  
        if peso_total > 3.0:  
            peso_extra = (peso_total - 3.0) *  
self.peso_extra_por_unidad  
        clima_extra = self.extras_clima.get(condicion_clima, 0.0)  
        return base + peso_extra + clima_extra
```

```
    def consume_por_mover(self, celdas: int = 1, peso_total: float =  
0.0, condicion_clima: str = "clear") -> float:  
        por_celda = self.consumo_por_celda_total(peso_total,  
condicion_clima)  
        cantidad = por_celda * max(0, int(celdas))  
        self._do_consume(cantidad)  
        return cantidad
```

```
    def _do_consume(self, cantidad: float):  
        self.resistencia -= cantidad  
        if self.resistencia <= 0.0:  
            self.resistencia = 0.0  
            self._exhaust_lock = True
```

```
    def recupera(self, segundos: float, rest_point: bool = False) ->  
float:  
        rate = self.recuperacion_rate_rest_point if rest_point else  
self.recuperacion_rate_idle  
        cant = rate * max(0.0, segundos)  
        self.resistencia += cant  
        if self.resistencia > self.resistencia_max:  
            self.resistencia = self.resistencia_max  
        if self._exhaust_lock and self.resistencia >=  
self.recuperacion_threshold:  
            self._exhaust_lock = False  
        return cant
```

```
    def estado_actual(self) -> Literal["normal", "cansado",  
"exhausto"]:  
        if self.resistencia <= 0.0:
```

```

        return "exhausto"
    elif self.resistencia <= self.recuperacion_threshold:
        return "cansado"
    else:
        return "normal"

    def factor_velocidad(self) -> float:
        est = self.estado_actual()
        if est == "normal":
            return 1.0
        elif est == "cansado":
            return 0.8
        else:
            return 0.0

    def puede_moverse(self) -> bool:
        if self._exhaust_lock:
            return self.resistencia >= self.recuperacion_threshold
        return self.resistencia > 0.0

    def to_dict(self) -> dict:
        return {
            "resistencia_max": self.resistencia_max,
            "resistencia": self.resistencia,
            "recuperacion_threshold": self.recuperacion_threshold,
            "exhaust_lock": self._exhaust_lock,
        }

    def load(self, d: dict):
        self.resistencia_max = float(d.get("resistencia_max",
self.resistencia_max))
        self.resistencia = float(d.get("resistencia", self.resistencia))
        self.recuperacion_threshold =
float(d.get("recuperacion_threshold", self.recuperacion_threshold))
        self._exhaust_lock = bool(d.get("exhaust_lock",
self._exhaust_lock))

```

Sigue sin detectar el `__post_init__`. Qué puedo hacer?

Lo copié como me diste, y me da esto: `AttributeError: type object 'Stats' has no attribute 'extras_clima'`

Al reproducirlo , la segunda linea me da como resultado `None`.

Resultado:

```

from dataclasses import dataclass, field
from typing import Literal, Dict

```

```

@dataclass
class Stats:
    resistencia_max: float = 100.0
    resistencia: float = 100.0

    recuperacion_threshold: float = 30.0
    consumo_por_celda: float = 0.5
    peso_extra_por_unidad: float = 0.2

    extras_clima: Dict[str, float] = field(default_factory=lambda: {
        "rain": 0.1,
        "wind": 0.1,
        "rain_light": 0.1,
        "storm": 0.3,
        "heat": 0.2,
        "clear": 0.0,
        "clouds": 0.0,
        "fog": 0.0,
        "cold": 0.0,
    })

    recuperacion_rate_idle: float = 5.0
    recuperacion_rate_rest_point: float = 10.0

    _exhaust_lock: bool = False

    def consumo_por_celda_total(self, peso_total: float,
    condicion_clima: str) -> float:
        base = self.consumo_por_celda
        peso_extra = 0.0
        if peso_total > 3.0:
            peso_extra = (peso_total - 3.0) *
self.peso_extra_por_unidad

        clima_extra = self.extras_clima.get(condicion_clima, 0.0)
        return base + peso_extra + clima_extra

```

Al tratar de reproducirlo, me da lo siguiente: ImportError: cannot import name 'Stats' from 'src.game.stats\_module'  
(c:\Users\Andrey\Documents\Programas Estructuras de  
datos\Proyecto-Datos\src\game\stats\_module.py

Ya dejó de dar ese error. Sin embargo, SIGUE SIN LEER  
EXTRAS\_CLIMA.

Me está dando este resultado: Archivo cargado:  
c:\Users\Andrey\Documents\Programas Estructuras de  
datos\Proyecto-Datos\src\game\stats\_module.py Código fuente de  
Stats.init: def init(self, resistencia\_max: float = 100.0,

```

recuperacion_threshold: float = 30.0, consumo_por_celda: float =
0.5, peso_extra_por_unidad: float = 0.2, recuperacion_rate_idle:
float = 5.0, recuperacion_rate_rest_point: float = 10.0): # atributos
base self.resistencia_max: float = resistencia_max self.resistencia:
float = resistencia_max self.recuperacion_threshold: float =
recuperacion_threshold self.consumo_por_celda: float =
consumo_por_celda self.peso_extra_por_unidad: float =
peso_extra_por_unidad # inicialización explícita, sin depender de
dataclass self.extras_clima: Dict[str, float] = { "rain": 0.1, "wind":
0.1, "rain_light": 0.1, "storm": 0.3, "heat": 0.2, "clear": 0.0,
"clouds": 0.0, "fog": 0.0, "cold": 0.0, } self.recuperacion_rate_idle:
float = recuperacion_rate_idle self.recuperacion_rate_rest_point:
float = recuperacion_rate_rest_point self._exhaust_lock: bool =
False
Traceback (most recent call last): File
"c:\Users\Andrey\Documents\Programas Estructuras de
datos\Proyecto-Datos\debug_stats.py", line 10, in <module>
print("\nInstancia creada -> extras_clima:", s.extras_clima)
^^^^^^^^^^^^^^^^ AttributeError: 'Stats' object has no attribute
'extras_clima'

```

Resultado:

class Stats:

```

def __init__(self, # 👉 así, con doble __
    resistencia_max: float = 100.0,
    recuperacion_threshold: float = 30.0,
    consumo_por_celda: float = 0.5,
    peso_extra_por_unidad: float = 0.2,
    recuperacion_rate_idle: float = 5.0,
    recuperacion_rate_rest_point: float = 10.0):
    self.resistencia_max: float = resistencia_max
    self.resistencia: float = resistencia_max
    self.recuperacion_threshold: float = recuperacion_threshold
    self.consumo_por_celda: float = consumo_por_celda
    self.peso_extra_por_unidad: float = peso_extra_por_unidad

    self.extras_clima: Dict[str, float] = {
        "rain": 0.1,
        "wind": 0.1,
        "rain_light": 0.1,
        "storm": 0.3,
        "heat": 0.2,
        "clear": 0.0,
        "clouds": 0.0,
        "fog": 0.0,
        "cold": 0.0,
    }

    self.recuperacion_rate_idle: float = recuperacion_rate_idle
    self.recuperacion_rate_rest_point: float =

```

```
recuperacion_rate_rest_point  
self._exhaust_lock: bool = False
```

Ya me sale lo esperado. Ahora, qué sigue para asegurar la funcionalidad?

Ahora sí, ayúdame a encontrar qué métodos puedo implementar en [reputation.py](#)

Resultado;  
Empieza en 70.

Sube con entregas exitosas, en especial si son puntuales o anticipadas.

Baja con retrasos.

Derrota automática si baja de 20.

Multiplicador Mrep: **reputación  $\geq 85$  da un bono en pagos (+5%).**

Penalización de tardanza: **reputación  $\geq 85$  puede mitigar al 50% la penalización de la primera entrega tardía del día.**

La reputación siempre está en rango [0, 100].

Listo! Quiero que hagas una recapitulación de todas las clases de game que ya tengo. Revisa que la clase reputation.py cumple con TODOS los requisitos del enunciado y cumple con la lógica y corrige en caso de que false algo. Una vez hecho eso, podemos proceder con las siguientes clases.

Ayúdame citando los métodos de [player.py](#) que debo hacer. Considera que yo tengo una carpeta que se llama sprites. En ella, tengo los sprites del player, con las 4 posiciones. Estos se llaman así: Spr\_delivery\_up.png, Spr\_delivery\_down.png, Spr\_delivery\_izq.png y Spr\_delivery\_der.png.

Resultado:

- Movimiento en mapa: con sprites hacia las 4 direcciones (**up, down, left, right**).
- Consumo de resistencia: integrado con **Stats**,



considerando peso y clima.

- Velocidad: afectada por estado (`normal=100%`, `cansado=80%`, `exhausto=0%`).
- Sprites: cambia automáticamente según la dirección.
- Reputación: ajusta al completar entregas, conectado con `Reputation`.
- Reset diario: para mitigar penalización de tardanza.
- Derrota: se puede chequear con `self.reputation.esta_derrotado()`.

Quisiera que me ayudes a comprobarlo en el [main.py](#).

El mapa que se muestra es demasiado grande. Quisiera que fuera la ventana del juego más pequeña. En dónde debo modificar eso? Considera todas las clases que hemos hecho, o si debe ser en una clase aparte.

Resultado:

```
WINDOW_WIDTH = 800
WINDOW_HEIGHT = 600
```

```
screen = pygame.display.set_mode((WINDOW_WIDTH,
WINDOW_HEIGHT))
```

Avanza con `weather_system.py`

Resultado:

```
import random
import time
from src.models.ResultadoClima import ResultadoClima
from src.models.ClimaData import ClimaData
from src.models.CondicionClima import CondicionClima
```

```
class WeatherSystem:
```

```
    """
```

Sistema de clima basado en cadena de Markov.

- Cambia el clima cada 45–60 segundos según probabilidades de transición.

- Se alimenta de datos cargados desde la API (`ResultadoClima / ClimaData`).

```
    """
```

```

def __init__(self, clima_data: ClimaData):
    # datos de transición desde el JSON
    self.clima_data: ClimaData = clima_data

    # estado actual
    self.current_condition: str = clima_data.initial.condition
    self.current_intensity: int = clima_data.initial.intensity

    # siguiente cambio programado
    self.next_change: float = time.time() +
self._tiempo_siguiente_cambio()

    # -----
    # Cadena de Markov
    # -----
    def _tiempo_siguiente_cambio(self) -> float:
        """Devuelve un número aleatorio entre 45 y 60 segundos para
el próximo cambio."""
        return random.randint(45, 60)

    def _proximo_estado(self, actual: str) -> str:
        """Calcula el próximo estado de clima según
probabilidades."""
        transiciones = self.clima_data.transition.get(actual, {})
        if not transiciones:
            return actual # sin transición definida

        estados = list(transiciones.keys())
        probabilidades = list(transiciones.values())
        return random.choices(estados, weights=probabilidades,
k=1)[0]

    # -----
    # Update loop
    # -----
    def update(self) -> None:
        """Debe llamarse en cada frame. Cambia el clima si llegó el
tiempo."""
        ahora = time.time()
        if ahora >= self.next_change:
            nuevo_estado =
self._proximo_estado(self.current_condition)

            # transición progresiva (ej: subir intensidad si cambia a
tormenta)
            if nuevo_estado == self.current_condition:
                # mismo estado →variar intensidad suavemente
                self.current_intensity = max(0, min(3,
self.current_intensity + random.choice([-1, 0, 1])))

```

```

else:
    # cambio de estado →resetear intensidad base
    self.current_condition = nuevo_estado
    self.current_intensity = 1

    # programar siguiente cambio
    self.next_change = ahora +
self._tiempo_siguiente_cambio()

# -----
# Info de estado
# -----
def get_condition(self) -> str:
    """Devuelve el nombre del clima actual (ej: 'rain', 'storm')."""
    return self.current_condition

def get_intensity(self) -> int:
    """Devuelve la intensidad actual del clima (0-3)."""
    return self.current_intensity

def get_effects(self) -> dict:
    """
    Devuelve los efectos en movimiento y resistencia:
    - speed_factor: multiplicador de velocidad.
    - resistance_penalty: consumo adicional.
    """
    condition = self.current_condition
    effects = {
        "speed_factor": 1.0,
        "resistance_penalty": 0.0
    }

    if condition in ["rain", "rain_light", "storm"]:
        effects["speed_factor"] = 0.9
        effects["resistance_penalty"] = 0.1
    elif condition == "wind":
        effects["speed_factor"] = 0.9
        effects["resistance_penalty"] = 0.1
    elif condition == "heat":
        effects["speed_factor"] = 0.8
        effects["resistance_penalty"] = 0.2
    elif condition == "cold":
        effects["speed_factor"] = 0.95
        effects["resistance_penalty"] = 0.05

    return effects

```

Quiero que me expliques de nuevo cada uno de los archivos que se crean en game, y que me digas qué propósitos tienen.

Necesito que leas el enunciado nuevamente, con las instrucciones específicas. ANALIZA TODO LO QUE HAS HECHO. Revisa el código de weather\_system.py: import time import random from src.models.ResultadoClima import ResultadoClima from src.models.ClimaData import ClimaData from src.models.CondicionClima import CondicionClima class WeatherSystem: def \_\_init\_\_(self, clima\_data: ClimaData): self.clima\_data = clima\_data self.condicion\_actual = clima\_data.initial.condition self.condicion\_intensidad = clima\_data.initial.intensity self.proximo\_cambio = time.time() + self.\_tiempo\_proximo\_cambio() def \_tiempo\_siguiente\_cambio(self): return random.randint(45, 60) def \_proximo\_estado(self, actual: str) -> str: transiciones = self.clima\_data.transition.get(actual, {}) if not transiciones: return actual #cuando no hay transiciones estados = list(transiciones.keys()) probabilidades = list(transiciones.values()) return random.choices(estados, weights=probabilidades, k=1)[0] def actualizar(self) -> None: ahora = time.time() if ahora >= self.proximo\_cambio: nuevo\_estado = self.\_proximo\_estado(self.condicion\_actual) if nuevo\_estado == self.condicion\_actual: self.condicion\_intensidad = max(0, min(3, self.condicion\_intensidad + random.choice([-1,0,1]))) else: self.condicion\_actual = nuevo\_estado self.condicion\_intensidad = 1 self.proximo\_cambio = ahora + self.\_tiempo\_siguiente\_cambio() def get\_condicion(self) -> str: return self.condicion\_actual def get\_intensidad(self) -> int: return self.condicion\_intensidad def get\_efectos(self) -> dict: condicion = self.condicion\_actual efectos = { "speed\_factor": 1.0, "resistance\_penalty": 0.0 } if condicion in ["rain", "rain\_light", "storm"]: efectos["speed\_factor"] = 0.9 efectos["resistance\_penalty"] = 0.1 elif condicion == "wind": efectos["speed\_factor"] = 0.9 efectos["resistance\_penalty"] = 0.1 elif condicion == "heat": efectos["speed\_factor"] = 0.8 efectos["resistance\_penalty"] = 0.2 elif condicion == "cold": efectos["speed\_factor"] = 0.95 efectos["resistance\_penalty"] = 0.05

ENGO ESTO. SOLO DAME EL RESTO: import time import random from typing import Dict, Any, Optional \_MULTIPLICADORES\_BASE: Dict[str, float] = { "clear": 1.00, "clouds": 0.98, "rain\_light": 0.90, "rain": 0.85, "storm": 0.75, "fog": 0.88, "wind": 0.92, "heat": 0.90, "cold": 0.92, } # Penalizaciones de resistencia base (alineado con Stats.extras\_clima) \_PENALIZACIONES\_BASE: Dict[str, float] = { "clear": 0.0, "clouds": 0.0, "rain\_light": 0.1, "rain": 0.1, "storm": 0.3, "fog": 0.0, "wind": 0.1, "heat": 0.2, "cold": 0.0, } class SistemaClima: def \_\_init\_\_(self, datos\_clima: Any, semilla: Optional[int] = None): """ datos\_clima debe contener: - initial.condition (str) - initial.intensity (float/int) en [0.0, 1.0] - transition: Dict[str, Dict[str, float]] """ if semilla is not None: random.seed(semilla) self.datos\_clima =

```

datos_clima # Condición inicial inicial_cond =
getattr(datos_clima.initial, "condition", "clear") inicial_int =
getattr(datos_clima.initial, "intensity", 1.0) self.condicion_actual:
str = str(inicial_cond) self.intensidad_actual: float = max(0.0,
min(1.0, float(inicial_int))) self.en_transicion: bool = False
self.transicion_inicio: float = 0.0 self.transicion_duracion: float =
0.0 self.condicion_origen: str = self.condicion_actual
self.condicion_destino: str = self.condicion_actual
self.intensidad_origen: float = self.intensidad_actual
self.intensidad_destino: float = self.intensidad_actual
self.mult_origen: float =
_MULTIPLICADORES_BASE.get(self.condicion_actual, 1.0)
self.mult_destino: float = self.mult_origen self.pen_origen: float =
_PENALIZACIONES_BASE.get(self.condicion_actual, 0.0)
self.pen_destino: float = self.pen_origen self.proximo_cambio:
float = time.time() + self._intervalo_siguiente() def
_intervalo_siguiente(self) -> float: return float(random.randint(45,
60)) def _duracion_transicion(self) -> float: return
random.uniform(3.0, 5.0)

```

Comprueba que esto corra bien:

```

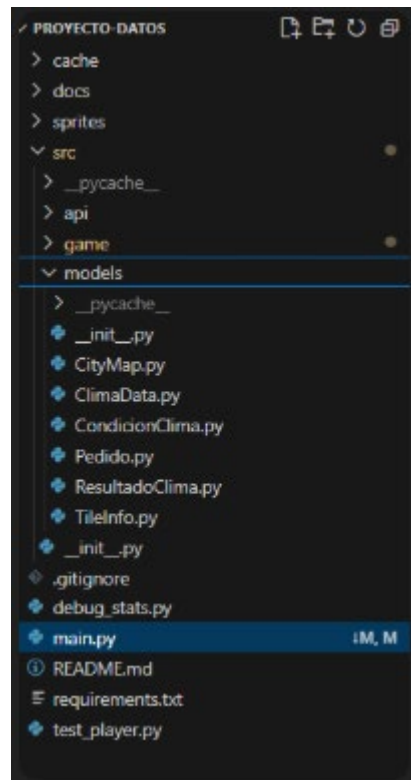
Import pygame import json from pathlib import Path from
src.models.CityMap import CityMap from src.game.map_render
import MapRenderer from src.game.stats_module import Stats
from src.game.reputation import Reputation from src.game.player
import Player def test_reputation(): print("=== PRUEBAS DE
REPUTATION ===") rep = Reputation() print("Inicial:", rep.valor,
rep.obtener_multiplicador_pago())
rep.registrar_entrega("a tiempo") print("Tras entrega a tiempo:",
rep.valor) rep.valor = 85 print("Bono aplicado?:",
rep.obtener_multiplicador_pago()) rep.registrar_entrega("tarde")
print("Tras tardanza mitigada:", rep.valor)
rep.registrar_entrega("tarde") print("Tras tardanza normal:",
rep.valor) rep.valor = 15 print("Derrota?:", rep.derrotado()) def
test_stats(): print("=== PRUEBAS DE STATS ===") s = Stats()
print("Inicial:", s.resistencia, s.estado_actual(), s.puede_moverse())
consumo = s.consume_por_mover(celdas=1, peso_total=2,
condicion_clima="clear") print("Mover 1 celda clear, peso 2 ->
consumo", consumo, "Resistencia:", s.resistencia) consumo =
s.consume_por_mover(celdas=1, peso_total=6,
condicion_clima="rain") print("Mover 1 celda lluvia, peso 6 ->
consumo", consumo, "Resistencia:", s.resistencia)
s.consume_por_mover(celdas=200, peso_total=1,
condicion_clima="storm") print("Tras tormenta:", s.resistencia,
s.estado_actual(), s.puede_moverse()) s.recupera(segundos=3,
rest_point=False) print("Tras 3s idle:", s.resistencia,
s.estado_actual(), s.puede_moverse()) s.recupera(segundos=2,
rest_point=True) print("Tras 2s rest:", s.resistencia,

```

```

s.estado_actual(), s.puede_moverse()) def main(): print("Iniciando
Courier Quest...") TILE_WIDTH = 40 TILE_HEIGHT = 44
BASE_DIR = Path(__file__).resolve().parent CACHE_DIR =
BASE_DIR / "cache" SPRITES_DIR = BASE_DIR / "sprites" #
cargar mapa with open(CACHE_DIR / "map.json", "r",
encoding="utf-8") as f: data = json.load(f) city_map =
CityMap(**data) print("Mapa cargado:", city_map.city_name,
city_map.width, city_map.height) pygame.init() font =
pygame.font.SysFont("Arial", 20) WINDOW_WIDTH = 800
WINDOW_HEIGHT = 600 screen = pygame.display.set_mode(
((WINDOW_WIDTH, WINDOW_HEIGHT)))
pygame.display.set_caption(city_map.city_name) clock =
pygame.time.Clock() renderer = MapRenderer(city_map,
SPRITES_DIR, TILE_WIDTH, TILE_HEIGHT) stats = Stats() rep
= Reputation() player = Player(SPRITES_DIR, stats, rep) # ahora
sí después del display print("Sprite inicial cargado:", player.image)
print("Rect inicial:", player.rect.topleft) player.mover("up")
print("Movido arriba:", player.rect.topleft) player.mover("right")
print("Movido derecha:", player.rect.topleft) running = True
print("Entrando al loop principal...") while running: for event in
pygame.event.get(): if event.type == pygame.QUIT: running =
False # input simple para mover al jugador keys =
pygame.key.get_pressed() moved = False if keys[pygame.K_UP]:
player.mover("up") moved = True elif keys[pygame.K_DOWN]:
player.mover("down") moved = True elif keys[pygame.K_LEFT]:
player.mover("izq") moved = True elif keys[pygame.K_RIGHT]:
player.mover("der") moved = True if not moved:
player.stats.recupera(segundos = 0.60, rest_point = False)
screen.fill((0, 0, 0)) renderer.draw(screen) screen.blit(player.image,
player.rect) hud_text = f"Resistencia: {player.stats.resistencia: .1f}
| Estado: {player.stats.estado_actual()} | Reputacion:
{player.reputation.valor}" hud_surface = font.render(hud_text,
True, (255, 255, 255)) screen.blit(hud_surface, (10,10))
pygame.display.flip() clock.tick(60) print("Saliendo del juego.")
pygame.quit() # ----- # EJECUCIÓN # -----
if __name__ == "__main__": test_reputation() test_stats() main()

```



La API se carga según la imagen, en models. La lógica que se carga del API es la siguiente: from typing import List, Dict from pydantic import BaseModel from .CondicionClima import CondicionClima class ClimaData(BaseModel): city: str initial: CondicionClima conditions: List[str] transition: Dict[str, Dict[str, float]]from pydantic import BaseModel class CondicionClima(BaseModel): condition: str intensity: intfrom pydantic import BaseModel from .ClimaData import ClimaData class ResultadoClima(BaseModel): version: str data: ClimaData

No olvides lo que hemos hecho hasta ahora. CONSIDERA EL PDF. Ok, haz la modificación de CondicionClima.py para tener todo de la forma más conveniente. Después, haz las modificaciones del weather\_system.py

Resultado:  
from pydantic import BaseModel, validator

```
class CondicionClima(BaseModel):
    condition: str
    intensity: float # siempre float en 0.0..1.0
```

```
@validator("intensity", pre=True)
def normalizar_intensidad(cls, v):
    """
```

Normaliza la intensidad:

- Si viene un entero (ej. 0..3), lo convierte a float 0.0..1.0
- Si viene un float ya entre 0.0..1.0, lo deja igual

```

"""
try:
    v = float(v)
    if v > 1.0: # probablemente API devuelva 0..3
        return max(0.0, min(1.0, v / 3.0))
    return max(0.0, min(1.0, v))
except Exception:
    return 1.0 # valor por defecto seguro

import time
import random
from typing import Dict, Any, Optional

_MUPLICADORES_BASE: Dict[str, float] = {
    "clear": 1.00,
    "clouds": 0.98,
    "rain_light": 0.90,
    "rain": 0.85,
    "storm": 0.75,
    "fog": 0.88,
    "wind": 0.92,
    "heat": 0.90,
    "cold": 0.92,
}
_PENALIZACIONES_BASE: Dict[str, float] = {
    "clear": 0.0,
    "clouds": 0.0,
    "rain_light": 0.1,
    "rain": 0.1,
    "storm": 0.3,
    "fog": 0.0,
    "wind": 0.1,
    "heat": 0.2,
    "cold": 0.0,
}

class SistemaClima:
    def __init__(self, datos_clima: Any, semilla: Optional[int] =
None):
        if semilla is not None:
            random.seed(semilla)

        self.datos_clima = datos_clima

        # Condición inicial
        inicial_cond = getattr(datos_clima.initial, "condition", "clear")
        inicial_int = getattr(datos_clima.initial, "intensity", 1.0)

```



```

self.condicion_actual: str = str(inicial_cond)
self.intensidad_actual: float = max(0.0, min(1.0,
float(inicial_int)))

# Transición
self.en_transicion: bool = False
self.transicion_inicio: float = 0.0
self.transicion_duracion: float = 0.0
self.condicion_origen: str = self.condicion_actual
self.condicion_destino: str = self.condicion_actual
self.intensidad_origen: float = self.intensidad_actual
self.intensidad_destino: float = self.intensidad_actual
self.mult_origen: float =
_MUPLICADORES_BASE.get(self.condicion_actual, 1.0)
self.mult_destino: float = self.mult_origen
self.pen_origen: float =
_PENALIZACIONES_BASE.get(self.condicion_actual, 0.0)
self.pen_destino: float = self.pen_origen
self.proximo_cambio: float = time.time() +
self._intervalo_siguiete()

def _intervalo_siguiete(self) -> float:
    return float(random.randint(45, 60))

def _duracion_transicion(self) -> float:
    return random.uniform(3.0, 5.0)

def _proximo_estado(self, actual: str) -> str:
    transiciones = getattr(self.datos_clima, "transition",
{ }).get(actual, { })
    if not transiciones:
        return actual
    estados = list(transiciones.keys())
    probabilidades = list(transiciones.values())
    return random.choices(estados, weights=probabilidades,
k=1)[0]

def actualizar(self) -> None:
    ahora = time.time()
    if self.en_transicion:
        progreso = min(1.0, (ahora - self.transicion_inicio) /
self.transicion_duracion)
        self.intensidad_actual = (1 - progreso) *
self.intensidad_origen + progreso * self.intensidad_destino
        # se pueden usar mult y pen si quieres debug o efectos
intermedios
        if progreso >= 1.0:
            self.en_transicion = False
            self.condicion_actual = self.condicion_destino

```

```

        self.intensidad_actual = self.intensidad_destino
        self.mult_origen = self.mult_destino
        self.pen_origen = self.pen_destino
        return

    if ahora >= self.proximo_cambio:
        nuevo = self._proximo_estado(self.condicion_actual)
        if nuevo == self.condicion_actual:
            self.intensidad_destino = max(0.0, min(1.0,
self.intensidad_actual + random.uniform(-0.2, 0.2)))
        else:
            self.en_transicion = True
            self.transicion_inicio = ahora
            self.transicion_duracion = self._duracion_transicion()
            self.condicion_origen = self.condicion_actual
            self.condicion_destino = nuevo
            self.intensidad_origen = self.intensidad_actual
            self.intensidad_destino = random.uniform(0.5, 1.0)
            self.mult_origen =
_MULTIPLICADORES_BASE.get(self.condicion_origen, 1.0)
            self.mult_destino =
_MULTIPLICADORES_BASE.get(self.condicion_destino, 1.0)
            self.pen_origen =
_PENALIZACIONES_BASE.get(self.condicion_origen, 0.0)
            self.pen_destino =
_PENALIZACIONES_BASE.get(self.condicion_destino, 0.0)
            self.proximo_cambio = ahora + self._intervalo_siguiente()

    def obtener_condicion(self) -> str:
        return self.condicion_actual

    def obtener_intensidad(self) -> float:
        return self.intensidad_actual

    def obtener_efectos(self) -> Dict[str, float]:
        """
        Devuelve efectos que afectan Stats:
        - factor_velocidad: multiplicador de velocidad
        - penalizacion_resistencia: penalización por clima
        """
        mult_base =
_MULTIPLICADORES_BASE.get(self.condicion_actual, 1.0)
        pen_base =
_PENALIZACIONES_BASE.get(self.condicion_actual, 0.0)

        return {
            "factor_velocidad": mult_base * self.intensidad_actual,
            "penalizacion_resistencia": pen_base *
self.intensidad_actual,

```

	<pre>         }  def tiempo_para_cambio(self) -&gt; float:     return max(0.0, self.proximo_cambio - time.time()) </pre>
deepseek	<p>te voy a dar un ejemplo de una situacion y quiero que me ayudes a pensar como implementarlo</p> <p>tengo un juego en desarrollo usando pygame, desde un API extraigo datos de paquetes que nuestro repartidor, el jugador, puede rechazar o aceptar, ocupo que en el juego mientras el jugador esta moviendo a el repartidor, entregando otros paquetes, o lo que este haciendo el jugador, cuando llegue un nuevo paquete desde el api, esta es la informacion del paquete desde la api:</p> <pre> {     "id": "PED-002",     "pickup": [         21,         18     ],     "dropoff": [         11,         17     ],     "payout": 260,     "deadline": "2025-09-01T12:09:00+00:00",     "weight": 2,     "priority": 1,     "release_time": 60 }, </pre> <p>ocupo que en el texto que aparecera en pantalla muestre, la id, payout, peso y prioridad</p> <p>dando la opcion al usuario de rechazarlo o aceptarlo, mediante un input del teclado z para aceptar y x para rechazar</p> <p>el texto que debera mostrar deberia ser algo como así en un rectangulo:</p> <p style="text-align: center;">Nuevo pedido!!!</p> <p>ID: PED-002   Pago: 260   peso: 2kg   Prioridad: 1</p> <p style="text-align: center;">(Z) aceptar                      (X) rechazar</p>
Copilot	"no funciona"

Chatgpt	<p>"Eres un asistente de desarrollo de videojuegos. Necesito implementar un HUD (Heads-Up Display) para un juego estilo Courier Quest usando Pygame. El HUD debe mostrar información en tiempo real sin bloquear el juego y debe incluir:</p> <p>Tiempo total transcurrido y tiempo restante de la misión.</p> <p>Nombre del jugador y score actual.</p> <p>Estado del jugador (resistencia, energía, o estado especial).</p> <p>Reputación y condiciones climáticas actuales.</p> <p>Cantidad de pedidos activos, pedidos pendientes y prioridad de cada pedido.</p> <p>Indicadores de inventario (qué paquetes están cargados y cuáles pendientes de entregar).</p> <p>Feedback visual cuando el jugador pueda recoger o entregar un pedido</p> <p>Actualizar dinámicamente cada frame y mantener compatibilidad con eventos de pausa.</p> <p>Genera un bloque de código en Python usando Pygame que implemente este HUD de manera modular y limpia, separando funciones de dibujo y cálculo de información. Añade comentarios explicativos para cada sección, incluyendo cómo actualizar la información en tiempo real y cómo integrar con el sistema de pedidos y el inventario del jugador."</p>
---------	---