| **IA usada** | **Prompt** |
| --- | --- |
| Copilot | 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.  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?  ¿Cuáles son las mejores prácticas para trabajar con APIs?  Conclusión: Usar pydantic para manejar los archivos JSON |
| Copilot | 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?  ¿Cómo puedo aplicarlo con la librería import junto con la librería pydantic? El link que quiero utilizar es: <https://tigerds-api.kindflower-ccaf48b6.eastus.azurecontainerapps.io>  ¿Y cómo yo sé qué tipo de parámetros maneja? Esto por medio de la lectura del archivo JSON? |
| 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?  ¿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"]  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"] |
| Copilot | 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. |
| ChatGPT | 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?  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?  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? |
| ChatGPT | 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?  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.  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?  ¿Hay alguna forma de hacer los sprites de score desde el mismo programa o no?  Me está dando un error: ModuleNotFoundError |
| ChatGPT | Estoy haciendo este código en Python usando Visual Studio Code (adjunto stats\_module.py). En él, el estado\_clima no es leido. 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.  \*Se adjunta el código de la resistencia\*  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?  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.  Me explicas qué es una cadena de markov? Cómo funciona en programación? De paso, dame algunos ejemplos de implementación de cadenas de markov.  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.  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  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). |
| ChatGPT | 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?  \*Se adjunta enunciado\*  Quiero que analices el enunciado. Para manejar toda la lógica de PyGame, quiero hacerlo en una carpeta aparte del main.py. ¿Es posible hacerlo?  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", "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 lo 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](http://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, climate\_condition="clear"). En él, da el siguiente error: got an unexpected keyword argument 'cells'  Resultado:  def consume\_for\_move(self, cells: int = 1, weight\_total: float = 0.0, climate\_condition: str = "clear") -> 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  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 <module> 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  Me da este archivo [stats.py](http://stats.py):  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):  # 👇 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, ayudame a encontrar qué métodos puedo implementar en [reputation.py](http://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 ≥ 85 da un bono en pagos (+5%).  **Penalización de tardanza**: reputación ≥ 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.  Ayudame citando los métodos de [player.py](http://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](http://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\_rend 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  \_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\_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 = \_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)  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,  }  def tiempo\_para\_cambio(self) -> float:  return max(0.0, self.proximo\_cambio - time.time()) |

Notas al readMe:

Debe instalarse la librería requests, pydantic y pygame.