Explicación para los Diagramas UML

Clase Juego

Atributos:

• **- jugador: Jugador** → El juego no puede existir sin un jugador humano.

• **- enemigo: Computadora** → Porque la computadora siempre persigue al jugador.

• **- laberinto: Laberinto** → El escenario del juego, sin él no habría muros ni pasillos.

• **- sonido: SistemaSonido** → Necesitamos reproducir efectos en eventos (mover, recolectar, captura).

• **- salon_fama: SalonFama** → Al terminar el juego se guarda el puntaje aquí.

• **- estado: str ('en curso', 'finalizado')** → Para saber si el juego está en curso o terminado.

Métodos:

• **+__ init__()** → Constructor que inicializa todos los componentes del juego.

• **+ iniciar(nombre: str): None** → Arranca el juego con un jugador, crea los objetos iniciales.

• **+ actualizar(): None** → Cada ciclo: mover jugador, mover enemigo, detectar colisiones, actualizar puntaje.

• **+ mostrar_estado(): None** → Debe mostrar cuántos puntos y vidas tiene el jugador (lo pide la especificación).

• **+ terminar(): None** → Cierra el juego cuando el jugador pierde todas las vidas y guarda en el Salón de la Fama.

• **+ salir(): None** → Maneja la confirmación y cierre ordenado de la aplicación.

Clase Personaje (abstracto)

Atributos:

• **- posicion: tuple[int, int]** → Coordenadas en el laberinto, común para jugador y computadora.

• **- velocidad: float** → Velocidad de movimiento, diferente para cada tipo de personaje.

Métodos:

• **+__init__()** → Constructor abstracto que inicializa posición y velocidad base. Las clases hijas deben llamar este constructor para heredar correctamente los atributos comunes.

• **+ mover(direccion: str): None** → Método que define el movimiento básico de cualquier personaje.

Propósito:

→ Clase abstracta que define comportamientos comunes entre Jugador y Computadora.

Clase Jugador *(hereda de Personaje)*

Atributos:

• **- nombre: str** → Lo pide la especificación: el jugador debe ingresar un nombre o alias para el Salón de la Fama.

• **- vidas: int = 3** → El jugador tiene exactamente 3 vidas.

• **- puntaje: int = 0** → El puntaje inicia en 0 y se incrementa con pasos y obsequios.

• **- posicion: tuple[int, int]** *(heredado de Personaje)* → Se necesita para saber dónde está en el laberinto.

Métodos:

• **+ __init()__** → Constructor que inicializa nombre, vidas y puntaje.

• **+ mover(direccion: str): None** → Para desplazarse con las flechas (↑ ↓ ← →).

• **+ sumar_puntos(puntos: int): None** → Suma puntos por pasos y obsequios.

• **+ perder_vida(): None** → Resta una vida cuando la computadora atrapa al jugador.

• **+ esta_vivo(): bool** → Indica si aún le quedan vidas.

Clase Computadora *(hereda de Personaje)*

Atributos:

• **- velocidad: float = 1.1** → Debe moverse un 10% más rápido que el jugador.

• **- posicion: tuple[int, int]** *(heredado de Personaje)* → Igual que el jugador, debe existir en el laberinto.

Métodos:

• **+ __init()__** → Constructor que inicializa posición y establece velocidad en 1.1.

• **+ perseguir(jugador: Jugador, laberinto: Labertinto): None** → Su comportamiento principal: moverse hacia el jugador evitando muros. • **+ mover(): None** → Cambia su posición en el laberinto.

Clase Administrador

Atributos:

• **- clave: str** → Necesaria para autenticar al administrador.

Métodos:

• **+ __init()__** → Constructor que establece la clave de acceso administrativo.

• **+ autenticar(clave_ingresada: str): bool** → Valida la clave secreta.

• **+ cargar_labertinto(archivo: str): Laberinto** → Permite seleccionar y cargar un laberinto desde archivo.

• **+ reiniciar_salon_fama(salon: SalonFama): None** → Limpia los registros del ranking.

Clase Laberinto

Atributos:

• **- muros: list[tuple[int, int]]** → Define dónde no se puede pasar.

• **- pasillos: list[tuple[int, int]]** → Define dónde sí se puede pasar.

• **- obsequios: list[Obsequio]** → Los ítems repartidos en el mapa.

Métodos:

• **+ __init()__** → Constructor que inicializa las listas de muros, pasillos y obsequios.

• **+ es_paso_valido(posicion: tuple[int, int]): bool** → Verifica si una posición no está bloqueada por un muro.

• **+ obtener_obsequio(posicion: tuple[int, int]): Obsequio | None** → Permite saber si en una celda hay un obsequio para recoger.

• **+ cargar_desde_archivo(archivo: str): None** → Para cargar la estructura desde un .json o .txt.

• **+ validar_estructura(datos: dict): bool** → Se exige una validación básica (posición inicial del jugador, enemigo, etc.).

Clase Obsequio

Atributos:

• **- posicion: tuple[int, int]** → Para saber en qué parte del laberinto está el obsequio.

• **- valor: int = 10** → Cada obsequio vale 10 puntos (según las reglas).

Métodos:

• **+ __init__()** → Constructor que inicializa posición y valor del obsequio.

• **+ recolectar(): int** → Retorna los puntos que aporta (10).

Clase SistemaSonido

Métodos:

• **+ __init__()** → Constructor que inicializa el sistema de sonido y prepara los recursos necesarios para reproducir efectos de audio.

• + **reproducir_movimiento(): None** → Sonido al mover al jugador.

• + **reproducir_obsequio(): None** → Sonido al recolectar un obsequio.

• + **reproducir_captura(): None** → Sonido cuando el enemigo atrapa al jugador.

Clase SalonFama

Atributos:

• **- registros: list[Registro]** → Para almacenar nombre, puntaje y laberinto.

• **- archivo: str** → Donde se guarda la información persistente (.txt o .json).

Métodos:

• **+ __init__()** → Constructor que inicializa la lista vacía de registros y establece la ruta del archivo donde se guardarán los puntajes de manera persistente.

• + **guardar_puntaje(registro: Registro): None** → Añade el puntaje del jugador actual.

• + **mostrar_mejores(): list[Registro]** → Devuelve los puntajes en orden descendente.

• + **cargar_datos(): None** → Carga los registros desde archivo al iniciar el programa.

• + **guardar_datos(): None** → Escribe los registros al archivo al actualizar.

• + **reiniciar(): None** → Borra todo el salón de la fama.

Clase Registro

Atributos:

• **- nombre_jugador: str** → Se pide mostrar el nombre en el ranking.

• **- puntaje: int** → Se debe mostrar y ordenar por él.

• **- laberinto: str** → El ranking debe mostrar también el laberinto usado.

Métodos:

• + **__init__(nombre_jugador: str, puntaje: int, laberinto: str)** → Constructor que inicializa todos los atributos del registro.