Huevos a la Fuga

En "Huevos a la Fuga", el jugador controla un huevo que ha caído de la nevera y descubre que la cocina es un lugar lleno de peligros. La misión del jugador es guiar al huevo esquivando obstáculos para así llegar a la ventana y escapar. El huevo cuenta con tres vidas representadas por su cáscara. Cada vez que choca contra un obstáculo, se forma una grieta debilitando su estructura. Si acumula tres grietas, el huevo se rompe por completo, convirtiéndose en un huevo revuelto y el juego termina.

Dinámica del Juego

El juego es una plataforma 2D con desplazamiento lateral donde el jugador debe moverse y saltar a través de la cocina esquivando peligros y utilizando power-ups para llegar a la meta.

- **Movimiento:** El huevo puede moverse a la izquierda, derecha y saltar.
- **Sistema de daño:** Cada vez que el huevo choca contra un obstáculo, se formarán grietas.Si acumula tres grietas, el huevo se rompe por completo, convirtiéndose en un huevo revuelto.

Niveles y Progresión

El juego tendrá tres niveles con dificultad creciente:

1. **Cocina Inicial:** El huevo cae de la nevera y aprende los controles básicos. Obstáculos simples como charcos de aceite.
2. **Zona de Cocción:** Se introducen sartenes calientes y utensilios en movimiento.
3. **Gran Escape:** La cocina se encuentra en total oscuridad, solo se ven los peligros, no las zonas seguras.

Obstáculos y Peligros

- **Sartenes calientes:** Contacto directo causa una derrota instantánea.
- **Charcos de aceite:** Provoca resbalones y movimientos incontrolables.
- **Luz apagada:** Oculta el camino. Solo brillan los sartenes y charcos.

Power-Ups

Para ayudar al jugador, el huevo puede recolectar power-ups:

- **Cáscara extra:** Reduce el daño de los choques por un tiempo limitado.
- **Turbo:** Aumenta la velocidad del huevo durante 3 segundos.
- **Papel de aluminio:** Hace al huevo invulnerable durante 3 segundos.

Condiciones de Victoria y Derrota

- **Victoria:** El huevo gana si logra llegar a la ventana y escapar.
- **Derrota:** El jugador pierde si:
    - El huevo se rompe completamente.
    - Cae en una sartén caliente.

Persistencia y Tabla de Clasificación

El juego guardará los resultados de cada partida:

- **Registro de puntajes:** Se guardará el tiempo que tarda el jugador en escapar.
- **Tabla de clasificación:** Mostrará los mejores tiempos de los jugadores.

Tecnologías y Desarrollo

- **PyGame**: Para la programación del juego, animaciones y detección de colisiones.
- **Sistema de físicas simple**: Para manejar la gravedad y los saltos.
- **Archivos JSON:** Para guardar el progreso y los puntajes de los jugadores.

Requisitos Funcionales

Identificación de requisitos:

1) Mover y controlar el personaje.
2) Detectar colisiones y aplicar daños.
3) Generar obstáculos y peligros.
4) Implementar power-ups.
5) Diseñar niveles y gestionar progresión.
6) Establecer condiciones de victoria y derrota.
7) Registrar puntajes y progreso.

Especificación y descomposición de requisitos:

Requisito 1:

Nombre	R1 - Mover y controlar el personaje
Resumen	El sistema permite que el huevo se desplace de manera horizontal (hacia los lados) y vertical (los saltos)
Entradas	- Tecla ← (movimiento izquierda). - Tecla → (movimiento derecha). - Tecla Espacio (salto). - Soltar teclas (detener movimiento)
Resultado	1. El huevo se desplaza horizontalmente o realiza un salto si está en el suelo. 2. Se aplican físicas básicas como gravedad, fricción, e interacción con el entorno (plataformas). 3. El sistema actualiza en tiempo real la posición del personaje y las animaciones correspondientes. 4. El personaje no puede atravesar obstáculos sólidos. 5. El movimiento se detiene al soltar las teclas.

Descomposición:

Pasos	Métodos	Responsables
+ posicion_x: float + posicion_y: float + velocidad: float + en suelo: bool	+init (self, posicion_x: float, posicion_y: float,velocidad: float)	Huevo

Mover personaje	mover_personaje(self, direccion: str)	Huevo
Realizar salto	saltar(self)	Huevo

Requisito 2:

Nombre	R2 - Detectar colisiones y aplicar daños
Resumen	El sistema permite detectar cuando el personaje choca con algún objeto y, si esto ocurre, aplicar un daño visible en forma de grietas.
Entradas	- Posición del personaje (coordenadas X, Y) - Posición y tipo de obstáculo - Estado actual del personaje (número de grietas: 0-3) - Velocidad del impacto
Resultado	1. Si colisiona con un obstáculo normal: se añade una grieta. 2. Si colisiona con una sartén caliente: se pierde la partida. 3. Si se acumulan tres grietas: el huevo se rompe y se pierde la partida. 4. Se reproduce efecto visual de la grieta en la posición correcta 5. Se actualiza la interfaz mostrando el estado del personaje

Descomposición:

Pasos	**Métodos**	**Responsables**
+ grietas: int + invulnerable: bool +tiempo_invulnerabilidad: float	detectar_colision(self, obstaculos: list)	sistema_colisiones
Aplicar daño	aplicar_dano(self, tipo_obstaculo: str)	Huevo
Verificar estado	verificar_estado_huevo(self)	Huevo

Requisito 3:

Nombre	R3 - Generar obstáculos y peligros
Resumen	El sistema permite incluir elementos en el juego que dificultan el camino del personaje, como superficies resbalosas, enemigos o trampas.
Entradas	- Nivel actual. - Información del mapa (dimensiones, layout).

	- Configuración de dificultad. - Posiciones ocupadas por otros elementos.
Resultado	1. Los obstáculos aparecen en posiciones predeterminadas según el diseño del nivel 2. Los obstáculos especiales tienen comportamientos únicos. 2.1 Charcos: causan deslizamiento y pérdida de control temporal. 2.2 Sartenes: causan muerte instantánea al contacto.. 3. Se valida que existe al menos un camino viable al objetivo. 4. Los obstáculos se posicionan respetando las reglas del nivel 5. Se inicializan las propiedades físicas y visuales de cada obstáculo.

Descomposición:

Pasos	Métodos	Responsables
+ tipo: str + posicion: tuple + efecto:str + activo: bool	init (self, tipo: str, posicion: tuple, efecto:str)	Obstaculo
Generar obstáculos	generar_obstaculos(self, nivel: int)	Gestor_niveles
Aplicar efecto	aplicar_efecto(self, personaje: Personaje)	Obstaculo

Requisito 4:

Nombre	R4 - Implementar power-ups
Resumen	El sistema permite agregar objetos especiales que el personaje puede recoger para obtener ventajas temporales, como moverse más rápido o resistir daños.
Entradas	- Posición del personaje. - Posición del power-up. - Tipo de power-up disponible. - Estado actual del personaje.
Resultado	1. El sistema detecta cuando el personaje colisionó con un power-up. 2. Se activa el efecto correspondiente según el tipo: 2.1 Cáscara extra: Reduce daño por tiempo limitado 2.2 Turbo:Aumenta velocidad por 3 segundos 2.3 Papel de aluminio: Invulnerabilidad por 3 segundos. 3. El power-up desaparece del mapa tras ser recolectado.

	4. Se muestra indicador visual del efecto activo. 5. El efecto se desactiva automáticamente al cumplir el tiempo.

Descomposición:

Pasos	Métodos	Responsables
+ tipo: str + duracion: float + activo:bool + tiempo_restante: float	init (self, tipo: str, posicion: tuple, duracion:float)	Power-up
Recolectar power-up	recolectar_powerup(self, powerup:PowerUp)	Power-up
Activar efecto	activar_efecto(self, personaje: Personaje)	Power-up

Requisito 5:

Nombre	R5 - Sistema de progresión por niveles
Resumen	El sistema permite organizar el juego en varios niveles, cada uno con mayor dificultad, para que el jugador avance poco a poco.
Entradas	- Nivel actual - Progreso del jugador - Dificultad ascendente - Estado del jugador (vida, posición)
Resultado	1. El sistema carga el nivel correspondiente con sus obstáculos específicos 2. Se ajusta la dificultad según el nivel. 3. Se valida que el jugador ha completado el nivel anterior. 4. Se desbloquea el siguiente nivel tras completar el actual.

Descomposición:

Pasos	Métodos	Responsables
+ numero: int + nombre: str + dificultad:str + completado: bool	init (self, numero: int, nombre: str,dificultad: str)	Nivel
Cargar nivel	cargar_nivel(self, numero: int)	Gestor_niveles
Verificar progresión	verificar_progresion(self, jugador: str)	Gestor_niveles

Requisito 6:

Nombre	R6 - Establecer condiciones de victoria y derrota
Resumen	El sistema permite definir cuándo el jugador gana (al llegar a la meta) o pierde (si el personaje se rompe, cae en una trampa o es atrapado).
Entradas	- Posición actual del jugador. - Estado del personaje (grietas, vida). - Posición de la meta (ventana). - Contacto con elementos letales
Resultado	1. El sistema verifica constantemente las condiciones de victoria. 1.1 El huevo llega a la ventana(meta) 2. El sistema verifica las condiciones de derrota. 2.1 El huevo acumula 3 grietas y se rompe. 2.2 El huevo toca una sartén caliente. 3. Se muestra la pantalla correspondiente(victoria/derrota). 4. Se registra el resultado de la partida. 5. Se ofrecen opciones para reiniciar o salir.

Descomposición:

Pasos	Métodos	Responsables
+ estado_juego: str + tiempo_partida:float + victoria: bool	verificar_condiciones(self)	Controlador_eventos
Finalizar partida	finalizar_partida(self, resultado: str, tiempo:float)	Controlador_eventos
Mostrar resultado	mostrar_resultado(self, tipo: str)	Interfaz de usuario

Requisito 7:

Nombre	R7 - Registrar puntajes y progreso
Resumen	El sistema permite guardar el tiempo que tardó el jugador en completar el nivel y mostrar una tabla con los mejores resultados.
Entradas	- Nombre del jugador - Tiempo de completitud - Nivel completado - Resultado de la partida
Resultado	1. El sistema guarda automáticamente el tiempo de cada partida

	completada. 2. Se actualiza la tabla de clasificación con los mejores tiempos. 3. Se muestra al jugador su posición en el ranking. 4. Los datos se almacenan en formato JSON para persistencia. 5. Se muestran estadísticas generales con los mejores tiempos.

Descomposición:

Pasos	Métodos	Responsables
+ nombre: str + tiempo: float + nivel:int + fecha: datetime	init (self, nombre: str, tiempo: float, nivel:int)	Puntuaciones
Guardar puntaje	guardar_puntaje(self, puntaje: Puntaje)	Gestor_puntaciones
Obtener ranking	obtener_ranking(self, nivel: int)	Gestor_puntaciones
Cargar datos	cargar_datos_json(self, archivo: str)	Gestor_puntaciones

Modelo del mundo

Clases principales:

1. Huevo (Personaje Principal)

Atributos:

- posicion_x: float - Coordenada horizontal
- posicion_y: float - Coordenada vertical
- velocidad: float - Velocidad de movimiento
- grietas: int - Número de grietas acumuladas (0-3)
- Vidas: int - Determina la cantidad de vidas que tiene

Métodos:

- __init__(posicion_x, posicion_y, velocidad)
- mover_personaje(direccion: str) - Mueve el huevo (izquierda/derecha)
- saltar() - Realiza un salto si está en el suelo
- aplicar_daño(tipo_obstaculo: str) - Aplica daño según el obstáculo
- aplicar_powerup(tipo_powerup) - Aplica el power up al huevo según su tipo
- actualizar(plataformas) - Permite aplicar por "fotograma" el movimiento del huevo

2. Obstáculo

Atributos:

- tipo: str - Tipo de obstáculo (sartén, charco, etc.)
- posicion: tuple - Coordenadas (x, y)
- efecto: str - Efecto que causa (daño, deslizamiento, muerte)
- activo: bool - Si está activo en el nivel

Métodos:

- __init__(tipo, posicion, efecto)
- aplicar_efecto(personaje: Huevo) - Aplica el efecto al personaje

Tipos de obstáculos:

- **Sartén caliente**: Muerte instantánea
- **Charco de aceite**: Deslizamiento y pérdida de control

3. PowerUps

Atributos:

- tipo: str - Tipo de power-up
- duracion: float - Duración del efecto en segundos
- activo: bool - Si está disponible para recoger
- tiempo_restante: float - Tiempo restante del efecto

Métodos:

- __init__(tipo, posicion, duracion)
- activar_efecto(personaje: Huevo) - Activa el efecto en el personaje

Tipos de power-ups:

- **Cáscara extra**: Reduce daño temporalmente
- **Turbo**: Aumenta velocidad por 3 segundos
- **Papel de aluminio**: Invulnerabilidad por 3 segundos

4. Nivel

Atributos:

- numero: int - Número del nivel (1-3)
- nombre: str - Nombre descriptivo del nivel
- dificultad: str - Nivel de dificultad (fácil, medio, difícil)
- completado: bool - Si el jugador ha completado el nivel
- obstaculos: list[Obstaculo] - Lista de obstáculos del nivel
- powerups: list[PowerUp] - Lista de power-ups disponibles
- meta_posicion: tuple - Posición de la ventana (meta)

Métodos:

- __init__(numero, nombre, dificultad)

Niveles del juego:

1. **Cocina Inicial**: Obstáculos simples, charcos de aceite
2. **Zona de Cocción**: Sartenes calientes y utensilios en movimiento
3. **Gran Escape**: Cocina en oscuridad, solo se ven los peligros

5. Puntaje

Atributos:

- nombre: str - Nombre del jugador
- tiempo: float - Tiempo de completitud en segundos
- nivel: int - Nivel completado
- fecha: datetime - Fecha y hora de la partida

Métodos:

- __init__(nombre, tiempo, nivel)

Clases Gestoras:

6. Gestor_Niveles

Responsabilidades:

- Cargar y configurar niveles
- Gestionar la progresión entre niveles
- Generar obstáculos según el nivel

Atributos:

- nivel_actual: int
- lista_niveles: list[int]
- obstáculos: list[Obstáculo]
- powerups: list [PowerUp]

Métodos:

- cargar_nivel(numero: int) - Carga el nivel especificado
- verificar_progresion(jugador: str) - Verifica si puede avanzar

7. Sistema_Colisiones

Responsabilidades:

- Detectar colisiones entre elementos

- Gestionar interacciones físicas

Métodos:

- detectar_colision(obstaculos: list) - Detecta colisiones con obstáculos
- detectar_colision_powerup(powerups: list) - Detecta recolección de power-ups
- detectar_meta(meta_posicion: tuple) - Detecta llegada a la meta

8. Controlador Evento

Atributos:

- estado_juego: str - Estado actual (jugando, victoria, derrota)
- tiempo_partida: float - Tiempo transcurrido en la partida

Métodos:

- procesar_eventos (estado_actual, nivel_actual, huevo, mostrar_puntuaciones, cargar_puntuaciones, callback) - Se encarga de procesar los eventos que van pasando durante el transcurso del juego
- procesar_movimiento - gestiona el movimiento horizontal del jugador.

9. Gestor_Puntaciones

Responsabilidades:

- Guardar y recuperar puntajes
- Gestionar la tabla de clasificación
- Persistencia en archivos JSON

Atributos:

- puntajes: list [Puntuacion]

Métodos:

- guardar_puntajes(puntaje: Puntaje) - Guarda un nuevo puntaje
- obtener_ranking(nivel: int) - Obtiene los mejores tiempos

10. Interfaz_Usuario

Responsabilidades:

- Mostrar información al jugador
- Renderizar elementos visuales
- Mostrar pantallas de victoria/derrota

Métodos:

- mostrar_resultado(tipo: str) - Muestra pantalla de resultado
- mostrar_grietas(cantidad: int) - Muestra estado del huevo
- mostrar_powerup_activo(tipo: str, tiempo: float) - Indica power-up activo

Diagrama de clases

https://drive.google.com/file/d/1UoEdVZ68YHUfsFRzJD53a-SHzbewBEOp/view?usp=sharing

Repositorio de GitHub

https://github.com/JoseM1715/APOO2---PROYECT.git