PROYECTO NEOSNAKE

INTEGRANTES

MATEO CARVAJAL ZAPATA

MARIANA HENAO ECHEVERRI

MARIA FERNANDA PIEDRAHITA MONTOYA

CURSO

ALGORITMOS Y PROGRAMACIÓN ORIENTADA A OBJETOS

PROFESOR

JESÚS ANDRÉS HINCAPIÉ LONDOÑO

Descripción del Problema.

Este proyecto tiene como objetivo rediseñar y mejorar el clásico juego Snake, incorporando una jugabilidad más dinámica, nuevas mecánicas, y un entorno visual más intuitivo. Se busca ofrecer una experiencia interactiva que combine desafío, adaptabilidad y control preciso por parte del jugador.

La lógica del juego se organiza a partir de clases que representan los elementos esenciales del mundo: la serpiente, sus movimientos, su crecimiento al alimentarse, la aparición de obstáculos, la gestión del tiempo y la puntuación en pantalla. El sistema debe permitir una interacción fluida y coherente entre estos elementos, garantizando una partida funcional y equilibrada.

Además, se incorporan diferentes niveles de dificultad que modifican la velocidad, los recursos disponibles y las condiciones del entorno, lo que amplía la rejugabilidad del juego. La interacción con el jugador se refuerza mediante una interfaz que muestra el estado de la partida en tiempo real, incluyendo el puntaje, los power-ups activos y el tiempo restante cuando aplique.

En conjunto, Neo-Snake propone una versión enriquecida del clásico, manteniendo su esencia pero apostando por una estructura más sólida, modular y escalable que permita futuras expansiones.

Modelo del mundo.

Identificación de entidades y características:

Con base en el enunciado se identifican las siguientes entidades (clases) y características (atributos):

Neo_Snake

- Dificultad
- Tiempo restante
- Nivel

Serpiente

- Cabeza
- Cuerpo
- Cola
- Segmentos
- Dirección
- Velocidad

Alimento

- Tipo
- Posición

Obstáculos

- Tipo
- Posición

PowerUp

- Tipo
- Posición
- Duración

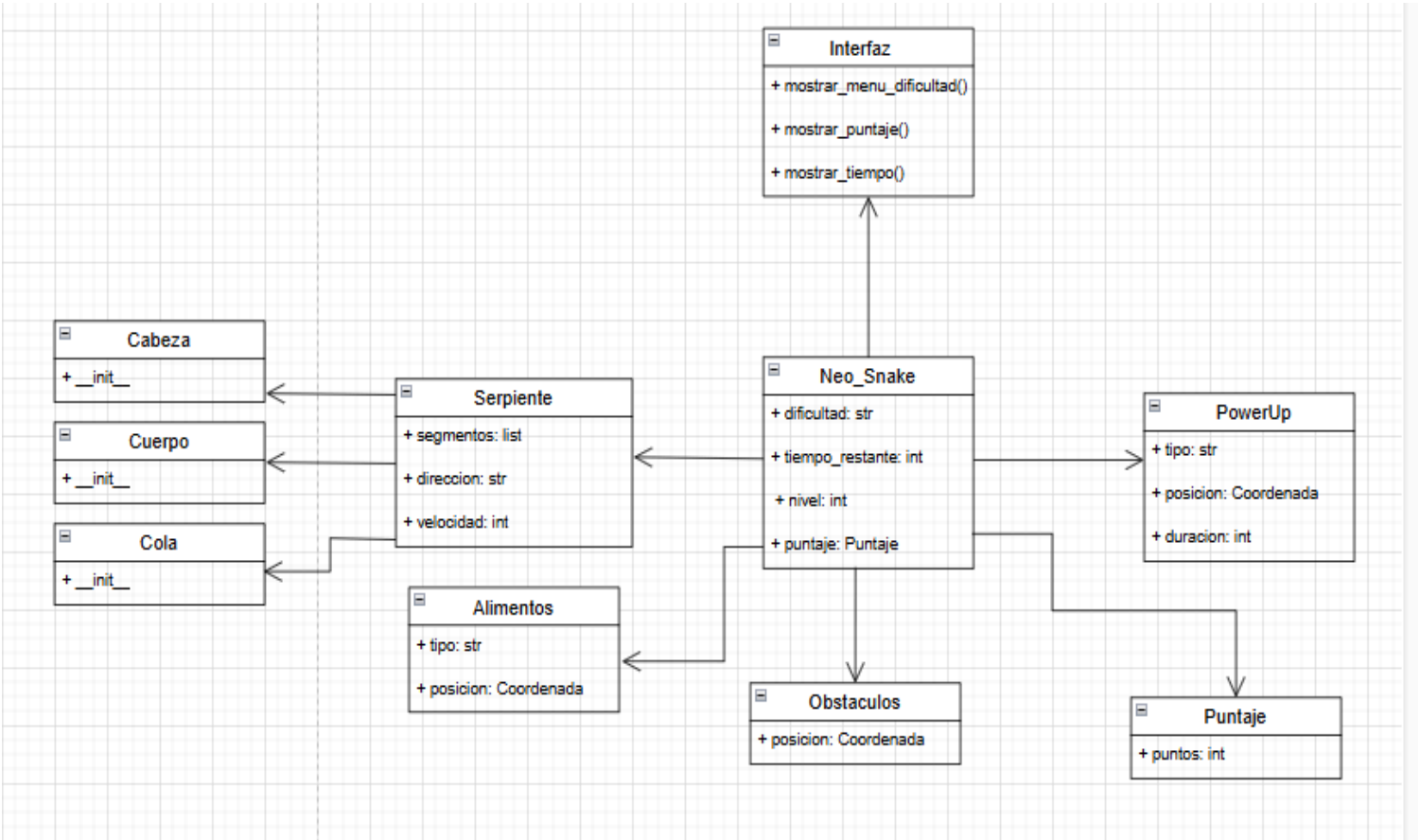
Puntaje

- Puntos

Interfaz

- Pantallas (menú, juego, fin)

2. Modelo de las clases



Requisitos Funcionales Neo-Snake.

Requisito 1 Movimiento de la Serpiente:

Nombre	R1- Movimiento de la Serpiente
Resumen	El sistema debe permitir que la serpiente se desplace en el mapa según la dirección seleccionada por el jugador, controlando de forma independiente la **`Cabeza`** y actualizando la posición del **`Cuerpo`** y la **`Cola`**.
Entradas	Dirección de movimiento (arriba, abajo, izquierda, derecha)
Resultados	1. La `Cabeza` de la serpiente se mueve en la dirección predefinida al inicio del juego. 2. El `Cuerpo` y la `Cola` siguen la trayectoria de la `Cabeza`. 3. Si el jugador cambia la dirección, la `cabeza` cambia su movimiento y el resto del cuerpo la sigue. 4. Si la serpiente colisiona con un obstáculo o consigo misma, se ejecuta el requisito R5 – Sistema de Colisiones.

Descomposición:

Pasos	**Métodos**	**Responsable**
Cambiar dirección	cambiar_direccion(nueva_direccion)	Cabeza
Mover cabeza	actualizar_posicion()	Cabeza
Actualizar cuerpo	actualizar_posicion()	Cuerpo
Actualizar cola	seguir_cuerpo()	Cola

Requisito 2 Crecimiento de la Serpiente al Comer:

Nombre	R2- Crecimiento de la Serpiente al Comer
Resumen	El sistema debe incrementar la longitud de la serpiente cuando ésta consume un alimento, añadiendo una nueva sección al `Cuerpo`.
Entradas	Colisión de la cabeza con el alimento.

Resultados	1. Si la cabeza de la serpiente toca un alimento, se elimina el alimento del mapa. 2. Se agrega una nueva sección al final del `Cuerpo`. 3. Se genera un nuevo alimento en una posición aleatoria del mapa. 4. Se incrementa el puntaje del jugador (**R3 – Sistema de Puntuación**).

Descomposición:

Pasos	**Métodos**	**Responsable**
Detectar colisión con alimento	colisionar_con_alimento()	Cabeza
Aumentar longitud del cuerpo	agregar_segmento()	Cuerpo
Ajustar posición de la cola	actualizar_posicion	Cola
Generar nuevo alimento	generar()	Alimento
Actualizar puntaje	actualizar_puntaje(puntos)	Puntaje

Requisito 3 Sistema de Puntuación:

Nombre	R3- Sistema de Puntuación
Resumen	El sistema debe registrar y actualizar el puntaje del jugador en función de su desempeño en la partida.
Entradas	Acción realizada por el jugador (comer alimento, usar power-up, sobrevivir por un tiempo determinado).
Resultados	1. Cada vez que la serpiente consume un alimento, el puntaje del jugador se incrementa en una cantidad fija. 2. Si el jugador recoge un power-up especial, se le otorgan puntos adicionales. 3. Si el jugador sobrevive durante un periodo determinado sin colisionar, recibe puntos extra. 4. Al finalizar la partida, el sistema guarda el puntaje en la base de datos si está dentro de los mejores registros. 5. Se muestra el puntaje en pantalla en tiempo real.

Descomposición:

Pasos	Métodos	Responsable
Incrementar puntaje al comer alimento	actualizar_puntaje(puntos: int)	Puntaje
Registrar puntaje al finalizar partida	guardar_puntaje()	Puntaje
Mostrar puntaje en pantalla	mostrar_puntaje()	Interfaz

Requisito 4 Modos de Dificultad:

Nombre	R4- Modos de Dificultad
Resumen	El sistema debe permitir al jugador elegir entre varios niveles de dificultad antes de iniciar una partida.
Entradas	Nivel de dificultad seleccionado (Fácil, Medio, Difícil, Extremo).
Resultados	**1.** El sistema muestra una pantalla con las opciones de dificultad. **2.** El jugador selecciona un nivel de dificultad **3.** Según el nivel elegido, se configuran los parámetros del juego: **3.1 Fácil:** Baja velocidad, pocos obstáculos, más alimentos, sin límite de tiempo. **3.2 Medio**: Velocidad media, obstáculos moderados, menos alimentos, tiempo límite de 5 minutos. **3.3 Difícil:** Velocidad alta, muchos obstáculos, escasez de alimentos, tiempo límite de 3 minutos. **3.4 Extremo:** Velocidad muy alta, gran cantidad de obstáculos, muy pocos alimentos, tiempo límite de 1 minuto. **4.** Se inicia la partida con los parámetros configurados.

Descomposición:

Pasos	Métodos	Responsable

Mostrar opciones de dificultad	mostrar_menu_dificultad()	Interfaz
Seleccionar dificultad	seleccionar_dificultad(nivel: str)	Juego
Configurar parámetros según dificultad	configurar_dificultad(nivel: str)	Juego
Iniciar partida con configuración seleccionada	iniciar_juego()	Juego

Requisito 5 Sistema de Colisiones:

Nombre	R5- Sistema de Colisiones
Resumen	El sistema debe detectar cuando la serpiente colisiona con los bordes, su propio cuerpo u obstáculos y actuar en consecuencia.niveles de dificultad antes de iniciar una partida.
Entradas	**1.** Posición de la Cabeza de la serpiente en el mapa **2.** Posiciones del Cuerpo de la serpiente **3.** Posición de los obstáculos en el mapa **4.** Configuración del modo de juego
Resultados	**1.** Si la cabeza choca con los bordes del mapa: **1.1** Se finaliza el juego. **1.2** Se muestra el puntaje obtenido. **2.** Si la cabeza choca con alguna parte del cuerpo: **2.1** Se finaliza el juego. **2.2** Se muestra el puntaje obtenido. **3.** Si la cabeza choca con un obstáculo: **3.1** Si el modo de dificultad permite atravesarlo con un power-up, no ocurre nada. **3.2** Si no tiene un power-up activo, se finaliza el juego. **3.3** Si la serpiente tiene un power-up de inmunidad activo, no sufre consecuencias por la colisión.

Descomposición:

Pasos	**Métodos**	**Responsable**
Detectar colisión con los bordes	verificar_colision_bordes() -> bool	Cabeza
Detectar colisión con el cuerpo	verificar_colision_cuerpo() -> bool	Cabeza

Detectar colisión con obstáculos	verificar_colision_obstaculo() -> bool	Cabeza
Aplicar consecuencia de colisión	manejar_colision()	Juego

Requisito 6 Generación de Elementos en el Mapa:

Nombre	R6- Generación de Elementos en el Mapa
Resumen	El sistema debe generar alimentos, obstáculos y power-ups en posiciones aleatorias según las reglas del juego.
Entradas	**1.** Tamaño del mapa **2.** Posición de la serpiente **3.** Nivel de dificultad
Resultados	**1.** Se genera un nuevo alimento cada vez que la serpiente consume uno. **2.** Los obstáculos se generan al inicio del juego según la dificultad. **3.** Los power-ups aparecen de forma aleatoria y desaparecen tras cierto tiempo si no son recogidos.

Descomposición:

Pasos	**Métodos**	**Responsable**
Generar alimento en una posición aleatoria	generar_alimento() -> Posición	Juego
Generar obstáculos según dificultad	generar_obstaculos() -> Lista de posiciones	Juego
Generar power-ups en el mapa	generar_powerups() -> Posición	Juego
Desaparecer power-up tras tiempo límite	eliminar_powerup()	Juego

Requisito 7 Control de Tiempo:

Nombre	R7- Control de Tiempo
Resumen	El sistema debe gestionar el tiempo de juego según el nivel de dificultad y mostrar una cuenta regresiva si aplica.

Entradas	Nivel de dificultad seleccionado
Resultados	**1.** Si el modo es **Fácil**, el juego no tiene límite de tiempo. **2.** Si el modo es **Medio**, la cuenta regresiva comienza en 5 minutos. **3.** Si el modo es **Difícil**, la cuenta regresiva comienza en 3 minutos. **4.** Si el modo es **Extremo**, la cuenta regresiva comienza en 1 minuto. **5.** Si el tiempo llega a 0, el juego finaliza y se muestra el puntaje obtenido.

Descomposición:

Pasos	**Métodos**	**Responsable**
Iniciar cuenta regresiva	iniciar_tiempo(duracion: int)	Juego
Reducir el tiempo progresivamente	actualizar_tiempo()	Juego
Finalizar el juego si el tiempo llega a 0	verificar_tiempo() -> bool	Juego

Requisito 8 Interfaz gráfica y visualización del puntaje:

Nombre	R8- Interfaz gráfica y visualización del puntaje
Resumen	El sistema debe mostrar información relevante al jugador durante la partida y al finalizar.
Entradas	**1.** Estado actual del juego **2.** Puntaje del jugador
Resultados	**1.** Durante la partida, se muestra en pantalla: **1.1** Puntuación actual **1.2** Tiempo restante (si aplica) **1.3** Power-ups activos **2.** Al finalizar la partida, se muestra un resumen con: **2.1** Puntaje final **2.2** Récord personal **2.3** Mejor puntuación registrada **2.4** Botón para reiniciar el juego o volver al menú

Descomposición:

Pasos	Métodos	Responsable
Mostrar puntaje en pantalla	actualizar_puntaje(puntos: int)	Juego
Mostrar tiempo restante	actualizar_tiempo_visual()	Juego
Mostrar power-ups activos	actualizar_powerups_visual()	Juego
Mostrar resumen final del juego	mostrar_resumen(puntaje: int)	Juego