**1. Resumen Ejecutivo y Finalidad del Proyecto**

**Resumen del Proyecto:**  
Este proyecto es una implementación moderna y completa del clásico videojuego "Snake", desarrollado en Java con la librería Swing para la interfaz gráfica. La aplicación no solo recrea la mecánica de juego principal, sino que también incorpora funcionalidades de una aplicación de escritorio robusta: un sistema de navegación multi-pantalla, registro de usuarios, persistencia de datos a través de una base de datos MS SQL Server, y un panel de reportes dinámico. La arquitectura del proyecto está rigurosamente basada en el patrón **Modelo-Vista-Controlador (MVC)** para garantizar un código limpio, modular y fácil de mantener.

**Finalidad y Objetivos Alcanzados:**  
La finalidad principal del proyecto fue aplicar y demostrar un dominio sólido de los principios de la **Programación Orientada a Objetos (POO)**. Los objetivos alcanzados incluyen:

* **Desarrollo de una Aplicación de Escritorio:** Creación de una GUI interactiva y estéticamente agradable.
* **Implementación de Lógica de Negocio:** La mecánica del juego (movimiento, crecimiento, colisiones) se encapsuló de forma efectiva.
* **Integración con Base de Datos:** Se logró una conexión exitosa con MS SQL Server para el registro de jugadores y el almacenamiento de puntajes, demostrando la capacidad de manejar la persistencia de datos.
* **Diseño de Software Avanzado:** La correcta implementación del patrón MVC y de componentes personalizados demuestra una comprensión profunda de la arquitectura de software.

**2. Arquitectura del Software: El Patrón Modelo-Vista-Controlador (MVC)**

La columna vertebral del proyecto es el patrón MVC. **Cada clase fue ubicada en un paquete específico (modelo, vista, controlador) según su única responsabilidad.** Esto es crucial porque previene que la lógica del juego se mezcle con el código de la interfaz, haciendo el proyecto mucho más fácil de entender y escalar.

* **Responsabilidad:** Contiene el estado del juego y las reglas de negocio. Es completamente independiente de la interfaz.
* **Clases y su Propósito:**
  + Serpiente, Comida, Tablero: Representan los objetos del juego. Serpiente maneja su propio cuerpo y movimiento; Comida sabe cómo generarse; y Tablero define los límites.
  + Punto, Direccion: Clases de soporte que mejoran la legibilidad y seguridad del código.
  + DatabaseManager: **Es el guardián de la base de datos.** Se ubica aquí porque la persistencia es una regla de negocio. Centraliza todas las consultas SQL, de modo que si quisiéramos cambiar a otra base de datos, solo esta clase necesitaría ser modificada.
* **Responsabilidad:** Mostrar la información del Modelo al usuario. No contiene lógica de juego.
* **Clases y su Propósito:**
  + GameFrame: El JFrame principal. Actúa como el **director de escena**, utilizando un CardLayout para cambiar entre los diferentes paneles (menús, juego, etc.). Es el único contenedor principal.
  + PanelJuego: El lienzo donde se dibuja el juego. Su método paintComponent es clave. **Aquí se implementaron las mejoras estéticas:**
    - Se dibuja una **cuadrícula de fondo** para dar un aspecto de tablero.
    - La **cabeza de la serpiente tiene un color distinto** para mejor visibilidad.
    - El **puntaje se dibuja directamente en la pantalla**, no en la barra de título, para una experiencia más inmersiva.
  + MainMenuPanel: La pantalla de bienvenida. Ha sido **rediseñada para ser visualmente impactante**, utilizando una imagen de fondo, un logo redimensionable y un componente de texto personalizado.
  + RegisterPanel, ReportsPanel: Paneles dedicados para el registro de usuarios y la visualización de reportes, manteniendo la interfaz organizada y cohesiva.
  + StrokedLabel: **Un componente personalizado que hereda de JLabel**. Se creó para resolver el problema de legibilidad del título en el menú. Sobrescribe paintComponent para dibujar un borde alrededor del texto, demostrando un uso avanzado de la herencia y el polimorfismo en Swing.
* **Responsabilidad:** Recibir la entrada del usuario (teclado, clics), actualizar el Modelo y notificar a la Vista que se redibuje.
* **Clase y su Propósito:**
  + ControladorJuego: Es el único controlador y el verdadero motor de la aplicación.
    - **Maneja el Timer del juego**, que define la velocidad y el ritmo del "game loop".
    - **Implementa KeyListener**, traduciendo las pulsaciones de teclas en cambios de dirección en el Modelo (Serpiente).
    - **Orquesta la comunicación:** Recibe la orden de "iniciar juego" desde la Vista (GameFrame), actualiza la Serpiente (Modelo) en cada "tick" del Timer, y luego le dice al PanelJuego (Vista) que se repinte (repaint()).

**3. Flujo de Funcionalidades Clave**

**1. Arranque y Navegación:**

* El main inicia el Controlador y el GameFrame.
* GameFrame inicializa todos sus paneles (Menú, Juego, etc.) y los añade a un CardLayout, mostrando el Menú por defecto.
* El usuario navega entre las pantallas haciendo clic en los botones. Cada clic en un botón del menú llama a un método en GameFrame (ej. mostrarPanel("REGISTRO")), que simplemente le dice al CardLayout qué "carta" mostrar.

**2. Inicio de una Partida:**

* En MainMenuPanel, el usuario introduce su pseudónimo y pulsa "Jugar".
* MainMenuPanel llama a gameFrame.iniciarJuego(pseudonimo).
* GameFrame hace dos cosas cruciales:
  1. Llama a controlador.iniciarNuevaPartida(pseudonimo) para que la lógica del juego se reinicie con el jugador correcto.
  2. Cambia la vista al PanelJuego y, muy importante, llama a panelJuego.requestFocusInWindow() para que el panel pueda "escuchar" el teclado.

**3. El Bucle del Juego y el Dibujado:**

* El Timer en ControladorJuego se dispara repetidamente.
* En cada disparo, el Modelo se actualiza (la serpiente se mueve).
* El Controlador llama a panelJuego.repaint().
* El método paintComponent de PanelJuego se activa. Este método **NO almacena una copia de la serpiente**, sino que en cada frame le pregunta al controlador: controlador.getSerpiente(). Esto asegura que siempre dibuja el estado más reciente del Modelo, evitando desincronizaciones.

**4. Fin de Partida y Persistencia:**

* Cuando el Controlador detecta una colisión, llama a finDelJuego().
* Este método detiene el Timer, obtiene la hora de fin y llama a dbManager.guardarPartida() con todos los datos.
* Muestra un JOptionPane al usuario y finalmente llama a gameFrame.mostrarPanel("MENU") para regresar al menú principal, completando el ciclo de juego.