Justificación de Diseño del Proyecto Final

# 1. Introducción

Este proyecto se desarrolló como parte del curso de Introducción a la Programación Orientada a Objetos,

con el objetivo de aplicar los conceptos vistos en clase a través de dos minijuegos: Ahorcado y Concentrece (parejas).

Se construyó una aplicación de consola con estructura modular, uso de clases, herencia, punteros, y archivos para guardar resultados.

La idea fue trabajar con recursos básicos del lenguaje C++ de forma clara y entendible.

# 2. Decisiones de diseño

# 2.1 Uso de clases y herencia

Se creó una clase abstracta llamada 'Juego', de la cual heredan las clases 'Ahorcado' y 'Parejas'. Esta decisión se tomó para:

- Evitar repetir atributos comunes como 'jugador', 'fecha', 'puntuación' y 'nombre del juego'.

- Aplicar el concepto de polimorfismo usando un método 'iniciar()' virtual puro que cada juego implementa a su manera.

Esto permitió tener un menú que maneja ambos juegos sin saber sus detalles internos.

# 2.2 Persistencia con archivos

Para guardar el historial de partidas se creó una clase 'GestorArchivo' con métodos para:

- Guardar partidas ('guardarPartida')

- Cargar, listar o eliminar historial

Esto permitió cumplir el requisito de mantener un registro por partida con formato:

[fecha] [jugador] [juego: AH/MEM] [resultado: G/P] [puntuación]

Cada juego llama a este gestor una vez termina para guardar su resultado.

# 2.3 Ahorcado

La clase 'Ahorcado' permite:

- Jugar contra otro jugador o contra la CPU

- Leer las palabras desde un archivo

- Mostrar visualmente el ahorcado parte por parte

El dibujo del ahorcado se hizo de forma modular, usando funciones como 'dibujarCabeza', 'dibujarCuerpo', etc.,

para mostrar el avance con cada fallo. Esto refleja el uso de condicionales básicos y refuerza la comprensión de estructuras sencillas.

# 2.4 Concentrece (Parejas)

La clase 'Parejas' permite:

- Cargar símbolos desde archivo (simbolos.txt)

- Mostrar el tablero con las posiciones numéricas

- Revelar las parejas descubiertas de forma persistente

- Limpiar la pantalla en cada turno para mejorar la visualización

El diseño fue hecho pensando en mantenerlo simple: el jugador solo necesita ingresar dos posiciones y ver cómo el tablero cambia.

Se usó un arreglo de símbolos mezclado al azar con Fisher-Yates sin std::shuffle, para evitar complejidad innecesaria.

# 2.5 Menú Principal

El menú fue hecho con opciones claras:

1. Jugar Ahorcado

2. Jugar Parejas

3. Ver historial

4. Eliminar historial

0. Salir

La clase 'MenuPrincipal' usa un puntero al 'GestorArchivo' y permite llamar al método 'iniciar()' de cualquier juego gracias al polimorfismo.

# 3. Estilo y simplicidad

Se evitó el uso de técnicas avanzadas como plantillas, punteros inteligentes o bibliotecas externas.

La intención fue demostrar comprensión real de los conceptos básicos de POO, más allá de copiar código complejo.

El diseño está orientado a reforzar:

- Claridad de clases

- Reutilización

- Separación de responsabilidades

- Trabajo con archivos planos

# 4. Conclusión

El proyecto logró cumplir todos los requisitos planteados: juegos funcionales, uso de clases abstractas, herencia, punteros, polimorfismo y persistencia.

Se priorizó la legibilidad y comprensión sobre lo técnico avanzado.

Cada decisión fue tomada buscando una solución coherente con el nivel del curso, mostrando que es posible hacer un proyecto completo con conceptos bien entendidos.