Break Mind {

De ="Walter Lozano - 2459487
"Sebastian Cardona - 2459714
"Alvaro Sanchez" - 2459482

}

Introducción {

Este proyecto implementa una aplicación de consola que incluye dos minijuegos: Ahorcado y Concentrese. El desarrollo se enfocó en demostrar el dominio de los conceptos fundamentales de la Programación Orientada a Objetos (POO), incluyendo herencia, polimorfismo, clases abstractas, composición, agregación y gestión dinámica de memoria mediante punteros.
La aplicación permite a los usuarios jugar ambos juegos de manera interactiva, guardar el historial de partidas y consultar estadísticas de juego, todo esto a través de una interfaz de consola amigable y funcional.

- Archivos del proyecto
- Implementacion conceptos POO
- Funcionalidades Implementadas
- Planificacion.

}

Archivos del Proyecto {

ARCHIVO	DESCRIPCION	RESPONSABILIDAD
Juego.h/cpp	**Clase abstracta base**	**Define la interfaz común y funcionalidad básica**
Ahorcado.h/cpp	Implementación del juego Ahorcado	Lógica específica del juego de adivinanza
Concentrese.h/cpp	Implementación del juego Concentrese	Lógica específica del juego de memoria
Menu.h/cpp	Interfaz de usuario	Gestión de menús y navegación
main.cpp	Punto de entrada	Inicialización y ejecución del programa

}

Implementacion De Conceptos POO {

Composición y Agregación

Composición: La clase Menu contiene un puntero a Juego, demostrando una relación de composición donde el menú "tiene-un" juego.

Beneficios:
Permite cambiar dinámicamente el tipo de juego
Desacopla la interfaz de usuario de la lógica de juego
Facilita la gestión de memoria

Polimorfismo

Implementación: Se utilizan métodos virtuales puros en la clase base y override en las clases derivadas.
Métodos polimórficos implementados:

iniciarJuego(): Configuración inicial específica de cada juego

jugar(): Lógica principal del juego

verificarFinJuego(): Condiciones de finalización

mostrarResultado(): Presentación de resultados

guardarPartida(): Persistencia personalizada

Herencia

Implementación: Las clases Ahorcado y Concentrese heredan de la clase abstracta Juego.
Beneficios:
Reutilización de código común (nombre del jugador, puntuación, persistencia)
Estructura consistente entre diferentes tipos de juegos
Facilita la extensión futura con nuevos juegos

Código ejemplo:
cpp

```cpp
class Ahorcado : public Juego {
    // Implementación específica del Ahorcado
};

class Concentrese : public Juego {
    // Implementación específica del Concentrese
};
```

Funcionalidades Implementadas {

Juego Concentrese

- **Tablero 4x4:** 16 posiciones con 8 parejas de símbolos
- **Interfaz numérica:** Posiciones numeradas del 1 al 16
- **Visualización dual:** Muestra tanto el estado actual como las posiciones numeradas
- **Algoritmo de mezcla:** Distribución aleatoria de las parejas
- **Puntuación:** Basada en la eficiencia (menos intentos = más puntos)
- **Validación:** Evita seleccionar posiciones ya reveladas

Juego del Ahorcado

- **Modo Jugador vs CPU:** Palabras predefinidas seleccionadas automáticamente
- **Modo Jugador vs Jugador:** Un jugador ingresa la palabra, otro la adivina
- **Control de intentos:** Máximo 6 intentos fallidos
- **Visualización:** Dibujo ASCII del ahorcado que se actualiza con cada error
- **Validación:** Evita repetir letras ya utilizadas
- **Puntuación:** Basada en la eficiencia (menos errores = más puntos)

}

Planificación {

FASE 1

- ✓ Definición del problema y objetivos del proyecto
- ✓ Identificación de requerimientos funcionales
- ✓ Diseño preliminar del sistema y estructura de clases
- ✓ Diagrama de clases con herencia, composición y agregación
- ✓ Selección de herramientas: lenguaje C++, compilador, editor

FASE 2

- ✓ Implementación de la clase abstracta Juego
- ✓ Definición de métodos virtuales puros y estructura base común
- ✓ Desarrollo de la clase Menu y lógica de navegación
- ✓ Creación del archivo main.cpp como punto de entrada
- ✓ Pruebas básicas de estructura sin lógica de juego aún

FASE 3

- ✓ Desarrollo completo del juego Ahorcado con modos y lógica
- ✓ Desarrollo del juego Concentrese con tablero dinámico
- ✓ Integración con el menú principal
- ✓ Aplicación de polimorfismo (jugar(), mostrarResultado(), etc.)
- ✓ Validación de entradas y control de errores

FASE 4

- ✓ Implementación del sistema de historial con archivo .txt
- ✓ Manejo de archivos mediante ifstream y ofstream
- ✓ Gestión de memoria dinámica con punteros (new/delete)
- ✓ Validación de punteros nulos
- ✓ Verificación de fugas de memoria

}