Informe Técnico: Hangman

Autor: Sebastian Blanco y Camilo Benavides

Fecha: Octubre 2025

1. Introducción

Este informe técnico presenta el desarrollo del juego del **ahorcado** implementado en el lenguaje de programación C++. El propósito del proyecto es diseñar una aplicación de consola que permita al usuario adivinar una palabra aleatoria, ingresando letras y evitando repeticiones, con un número limitado de vidas.

El ejercicio busca reforzar los conocimientos en programación estructurada, manejo de cadenas y validación de entradas de usuario, al tiempo que fomenta el diseño modular y el uso de funciones auxiliares.

2. Diseño de la Solución

2.1. Estructuras de datos utilizadas

- **Arreglo de caracteres** (`char almacen[26]`): almacena las letras ya utilizadas para evitar repeticiones.
- **Arreglo de cadenas** (`string palabras[]`): contiene las palabras posibles del juego.
- **Variables tipo** `string`: se usan para la palabra seleccionada, la versión descubierta y la entrada del usuario.

2.2. Justificación del uso de arreglos de caracteres

El arreglo de caracteres es eficiente y de fácil acceso mediante índices. Permite comprobar rápidamente si una letra ya ha sido usada, sin necesidad de estructuras dinámicas más complejas. Dado que solo se manejan 26 posibles letras, un tamaño fijo es suficiente y óptimo para este propósito.

2.3. Flujo principal del programa

El programa presenta un menú con dos opciones: **jugar** o **salir**. Al elegir la opción de juego:

1. Se elige una palabra aleatoria del arreglo `palabras[]`.
2. Se inicializa una cadena con guiones bajos para mostrar el progreso del jugador.
3. El usuario ingresa letras, las cuales se validan.
4. Si la letra es correcta, se muestra en su posición; de lo contrario, se resta una vida.
5. El ciclo termina cuando se adivina la palabra o se agotan las vidas.

3. Implementación

3.1. Algoritmo de verificación de letras

El algoritmo recorre la palabra, comparando cada carácter con la letra ingresada. Si hay coincidencia, se reemplaza el guion por la letra correspondiente.

```
bool acierto = false;
for (size_t i = 0; i < palabra.size(); i++) {
    if (palabra[i] == c) {
        descubierta[i] = c;
        acierto = true;
    }
}
if (!acierto) vidas--;
```

3.2. Manejo de la validación de entrada

Para evitar errores, se comprueba que la entrada:

- No sea vacía ni contenga espacios.
- Sea una sola letra.
- Corresponda a un carácter alfabético.

```
if (entrada.length() != 1 || !isalpha(entrada[0])) {
    cout << "Entrada invalida. Solo puedes ingresar letras." << endl;
    continue;
}
```

3.3. Estrategia para evitar letras repetidas

Se recorre el arreglo `almacen` para verificar si la letra ya fue usada:

```
bool yaUsada = false;
for(size_t j = 0; j < 26; j++){
    if (almacen[j] == c){
        cout << "Esta letra ya la pusiste, intenta con otra." << endl;
        yaUsada = true;
        break;
    }
}
if (yaUsada) continue;
```

4. Pruebas

Para comprobar el correcto funcionamiento del programa, se realizaron tres ejecuciones representativas. Cada una ilustra un escenario distinto: victoria, derrota y validación de entradas no válidas. Las evidencias gráficas correspondientes se presentan en el **Anexo**.

Caso 1: Juego ganado

En este escenario, el jugador logra adivinar correctamente la palabra **TRABAJO** antes de quedarse sin vidas. El sistema muestra los mensajes de acierto y el número de vidas restantes, finalizando con el texto **"¡Ganaste! La palabra era: TRABAJO"**. Este caso demuestra que la lógica de actualización y verificación de letras funciona de forma adecuada.

Caso 2: Juego perdido

Aquí, el jugador no logra adivinar la palabra antes de que las vidas lleguen a cero. El programa informa en cada intento la cantidad de vidas restantes y, al finalizar, muestra el mensaje **"Perdiste! La palabra era: JUEGO"**. Este resultado confirma el correcto control de condiciones de derrota.

Caso 3: Validación de entradas

En esta ejecución, se evalúa la robustez del sistema ante entradas inválidas. El usuario ingresa números, símbolos y cadenas largas, y el programa responde con mensajes como **"Entrada inválida. Solo puedes ingresar letras"** o **"Error: Debes ingresar solo**

UNA letra". Esto evidencia un control adecuado de la entrada del usuario, garantizando estabilidad y claridad en la interacción.

En conclusión, las pruebas realizadas validan el funcionamiento completo del programa en los tres posibles resultados de ejecución.

Conclusiones

El desarrollo del juego permitió reforzar el uso de estructuras básicas de control, validación de datos y manipulación de cadenas en C++. El principal desafío fue implementar una verificación robusta de la entrada del usuario y el control de letras repetidas, solucionado con el uso del arreglo `almacen`.

El código resultante es modular, eficiente y cumple con los objetivos planteados. Como mejora futura, se podrían incorporar niveles de dificultad, registro de puntuaciones y una interfaz más interactiva.

Anexo: Evidencias de Ejecución

A continuación se presentan las capturas correspondientes a los tres casos descritos en la sección de pruebas.

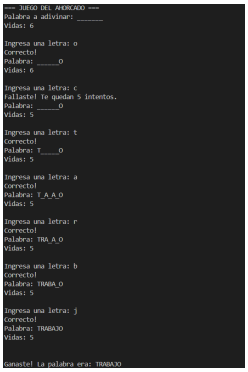

Figura 1: Caso 1: Juego ganado. Palabra adivinada correctamente.

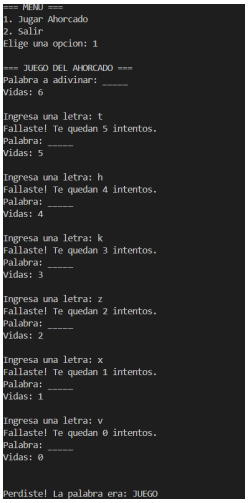

Figura 2: Caso 2: Juego perdido. El jugador agota sus vidas.

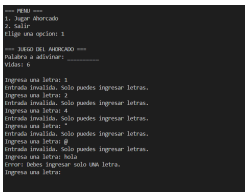

Figura 3: Caso 3: Validación de entradas no válidas.