****Universidad Internacional del Ecuador

**Escuela de Ingeniería en Sistemas de información**

[LOGICA DE PROGRAMACION 2-ECC-1A](https://uide.instructure.com/courses/25520)

**Aprendizaje Autónomo 2**

**[MIGUEL ANGEL ORTEGA PINEDA]**

Docente: PhD. MONICA PATRICIA SALAZAR TAPIA

Quito, 24 de agosto del 2025

* **Cual fue el análisis realizado y una explicación del funcionamiento esperado del sistema en función de sus diagramas funcionales.**

A continuación, se presenta un análisis del juego "El Ahorcado" y una explicación de su funcionamiento esperado, asumiendo la existencia de diagramas funcionales que no han sido provistos directamente en la solicitud, pero cuya estructura general se infiere para la explicación.

El Ahorcado: Análisis y Funcionamiento del Sistema

"El Ahorcado" es un juego de adivinanzas de palabras simple pero clásico, donde un jugador intenta adivinar una palabra secreta seleccionando letras. Por cada letra incorrecta, se dibuja una parte de una figura de "ahorcado". El juego termina cuando el jugador adivina la palabra (ganar) o se completa la figura del ahorcado (perder).

**1. Análisis del Juego "El Ahorcado"**

**Para un análisis detallado, consideramos los siguientes aspectos:**

**1.1. Componentes Clave:**

* **Palabra Secreta:** La palabra a adivinar. Debe ser seleccionada de un diccionario o lista predefinida.
* **Letras Adivinadas:** Las letras que el jugador ya ha intentado, tanto correctas como incorrectas.
* **Intentos Restantes/Partes del Ahorcado:** Un contador o indicador visual de cuántas oportunidades le quedan al jugador antes de perder.
* **Interfaz de Usuario (UI):**
  + **Representación de la Palabra:** Generalmente guiones bajos para las letras no adivinadas y las letras correctas en sus posiciones.
  + **Teclado/Selector de Letras:** Una forma para que el jugador ingrese o seleccione letras.
  + **Figura del Ahorcado:** Representación visual del progreso del "ahorcado".
  + **Mensajes de Estado:** Indicaciones de "Ganaste", "Perdiste", "Letra incorrecta", "Ya usaste esa letra", etc.
* **Diccionario de Palabras:** Un conjunto de palabras válidas de las cuales se selecciona la palabra secreta.

**1.2. Reglas del Juego:**

* **Inicio:** Se selecciona una palabra secreta y se muestra su representación inicial (solo guiones).
* **Turnos:** El jugador selecciona una letra por turno.
* **Letra Correcta:** Si la letra está en la palabra secreta, se revela en todas sus posiciones.
* **Letra Incorrecta:** Si la letra no está en la palabra secreta, se cuenta como un intento fallido y se dibuja una parte del ahorcado.
* **Letra Repetida:** Si el jugador intenta una letra que ya fue usada, no se penaliza (no se cuenta como intento fallido), pero se informa al jugador.
* **Condición de Victoria:** El jugador gana si adivina todas las letras de la palabra antes de agotar los intentos.
* **Condición de Derrota:** El jugador pierde si se agotan todos los intentos (la figura del ahorcado está completa) antes de adivinar la palabra.

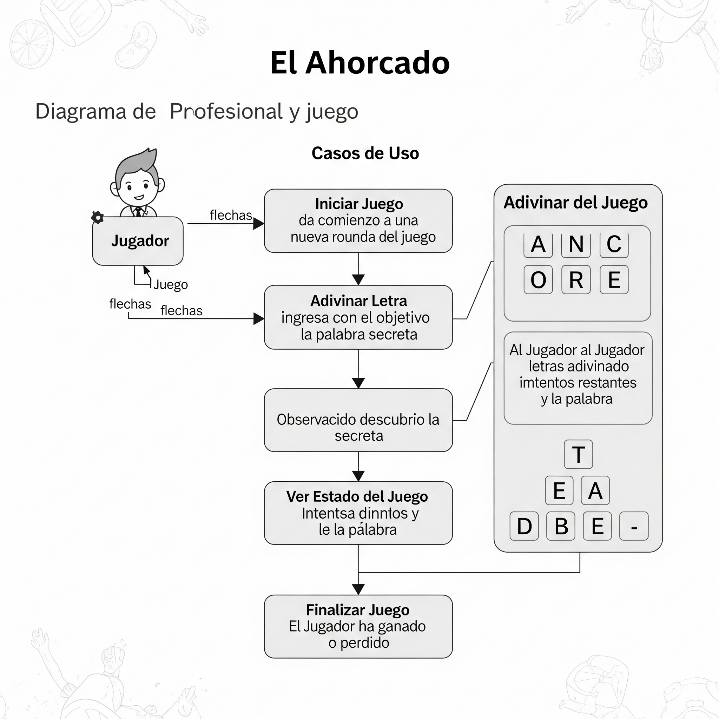
**1.3. Posibles Extensiones/Consideraciones Adicionales:**

* **Niveles de Dificultad:** Variar la longitud de la palabra o el número de intentos permitidos.
* **Categorías de Palabras:** Permitir al usuario elegir una categoría (animales, países, etc.).
* **Multijugador:** Permitir que dos jugadores jueguen (uno elige la palabra, el otro adivina).
* **Persistencia:** Guardar puntuaciones altas o el progreso del juego.

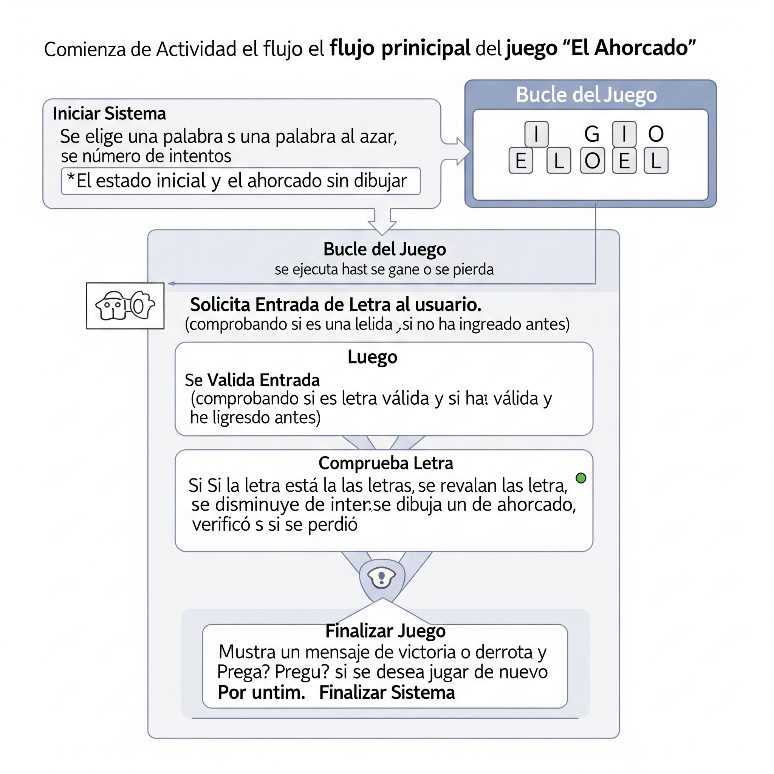
2. Funcionamiento Esperado del Sistema en Función de sus Diagramas Funcionales (Hipótesis)

Asumiendo que los "diagramas funcionales" se refieren a diagramas de flujo de datos (DFD), diagramas de caso de uso, o diagramas de actividad que modelan los procesos y la interacción, la explicación se centrará en los flujos lógicos y las interacciones.

**2.1. Diagrama de Caso de Uso (Ejemplo Implícito):**



* **Actor Principal:** Jugador
* **Casos de Uso Principales:**
  + **Iniciar Juego:** El jugador inicia una nueva partida.
  + **Adivinar Letra:** El jugador intenta adivinar una letra.
  + **Ver Estado del Juego:** El jugador observa el progreso de la palabra y el ahorcado.
  + **Finalizar Juego:** El juego termina por victoria o derrota.

**2.2. Diagrama de Actividad/Flujo de Procesos (Ejemplo Implícito):**

El funcionamiento del sistema seguiría un flujo similar al siguiente, que podría ser representado en un diagrama de actividad:

1. **Inicio del Sistema:**
   * Se carga la interfaz de usuario.
   * Se presenta la opción de "Iniciar Juego".
2. **Iniciar Juego (Subproceso):**
   * El sistema selecciona una palabra aleatoria del diccionario.
   * Inicializa el contador de intentos (ej. 6 o 7).
   * Crea la representación de la palabra (ej. \_ \_ \_ \_ \_).
   * Borra las letras adivinadas previamente.
   * Muestra el estado inicial al jugador.
3. **Bucle Principal del Juego (Mientras no haya Ganador ni Perdedor):**
   * **Solicitar Entrada de Letra:** El sistema espera que el jugador ingrese o seleccione una letra.
   * **Validar Entrada:**
     + ¿Es una letra válida? (ej. No números o símbolos).
     + ¿Ya fue intentada esa letra?
       - Si sí: Mostrar mensaje "Ya usaste esa letra" y volver a solicitar entrada.
       - Si no:
         * Añadir la letra a la lista de letras adivinadas.
         * **Comprobar Letra:**

**Si la letra está en la palabra secreta:**

Revelar la(s) posición(es) de la letra en la palabra mostrada.

Mostrar mensaje "¡Correcto!".

**Comprobar Victoria:** ¿Todas las letras de la palabra han sido adivinadas?

Si sí: Ir a "Finalizar Juego (Ganar)".

Si no: Volver a "Solicitar Entrada de Letra".

**Si la letra NO está en la palabra secreta:**

Incrementar el contador de partes del ahorcado dibujadas (o decrementar intentos restantes).

Mostrar mensaje "Incorrecto. ¡Intenta de nuevo!".

Actualizar la representación visual del ahorcado.

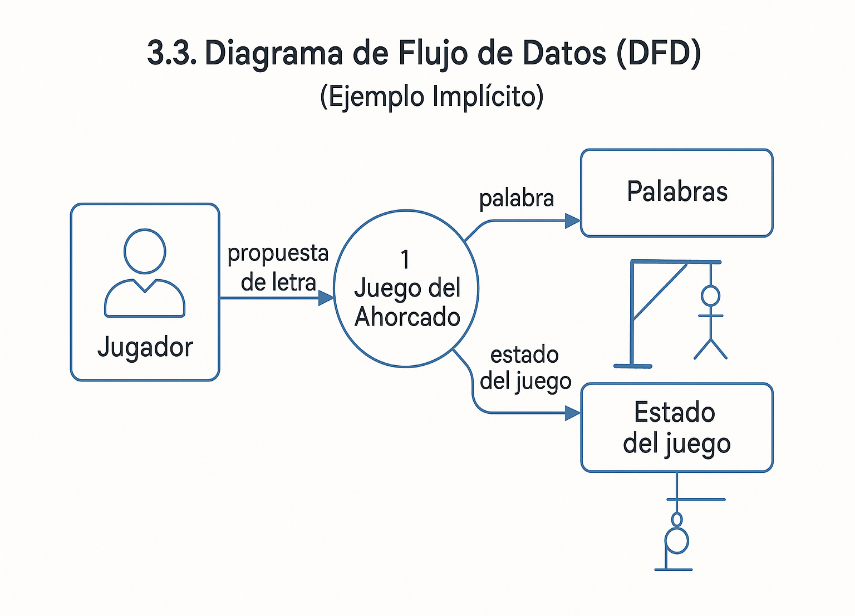
**Comprobar Derrota:** ¿Se han agotado todos los intentos?

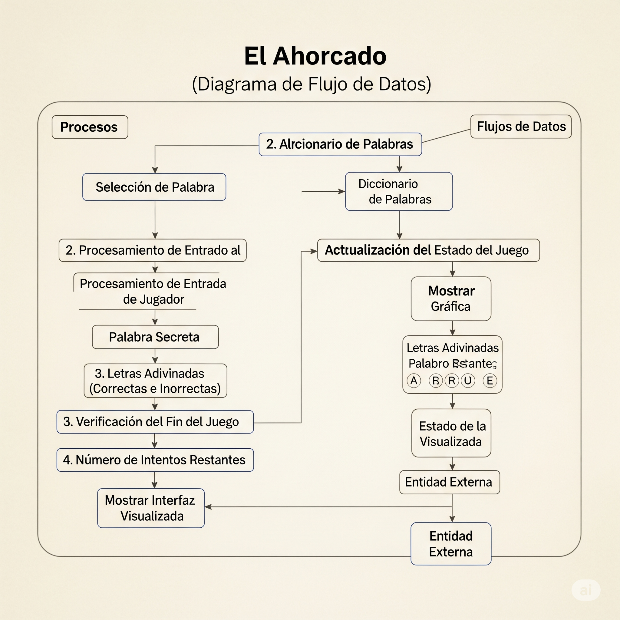
Si sí: Ir a "Finalizar Juego (Perder)".

Si no: Volver a "Solicitar Entrada de Letra".

1. **Finalizar Juego (Subproceso):**
   * **Si el jugador ganó:**
     + Mostrar mensaje "¡Felicidades, ganaste! La palabra era [Palabra Secreta]".
     + Ofrecer opción de "Jugar de Nuevo".
   * **Si el jugador perdió:**
     + Mostrar mensaje "Perdiste. La palabra era [Palabra Secreta]".
     + Mostrar la figura del ahorcado completa.
     + Ofrecer opción de "Jugar de Nuevo".
2. **Fin del Sistema:**
   * Si el jugador decide no jugar de nuevo, el sistema se cierra o vuelve a la pantalla inicial.

2.3. Diagrama de Flujo de Datos (DFD) (Ejemplo Implícito): imagen toda en español del juego

****del Ahorcado



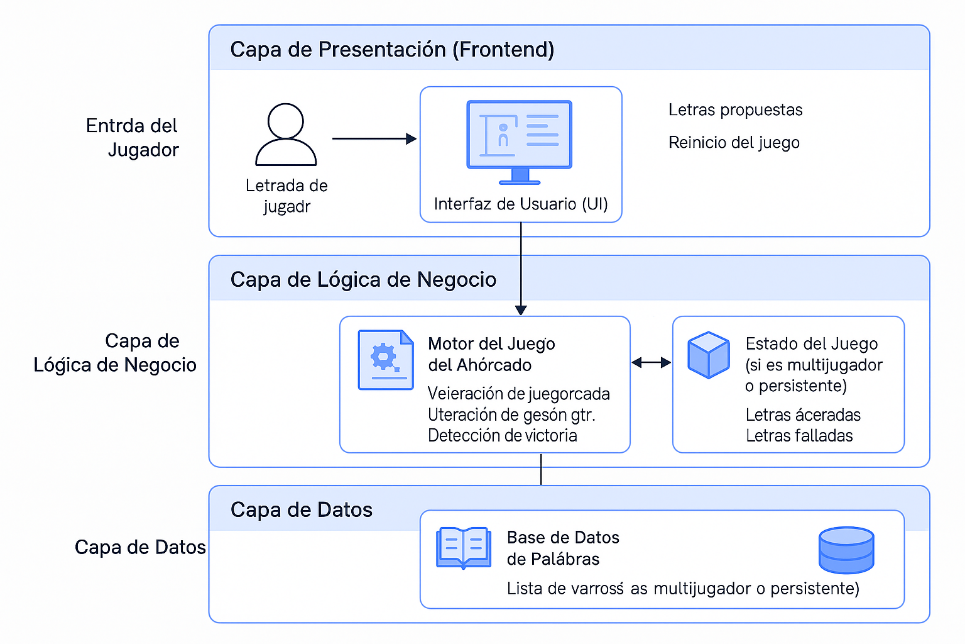
1. Entidades Externas:

- Jugador: La persona que juega.  
- Sistema: El software o aplicación que gestiona el juego.

2. Procesos:  
- Iniciar Juego: El jugador inicia el juego.  
- Seleccionar Palabra: El sistema selecciona una palabra aleatoria.  
- Mostrar Palabra: El sistema muestra la representación de la palabra con guiones bajos por cada letra.  
- Solicitar Letra: El sistema pide al jugador que ingrese una letra.  
- Verificar Letra: El sistema verifica si la letra ingresada está en la palabra.  
- Actualizar Estado: El sistema actualiza el estado del juego (palabra revelada, intentos restantes).  
- Mostrar Resultados: El sistema muestra el resultado si el jugador gana o pierde.  
  
3. Almacenes de Datos:  
- Lista de Palabras: Un almacenamiento donde están guardadas las palabras que se pueden seleccionar.  
- Historial de Jugadas: Información sobre las jugadas anteriores (puede incluir palabras adivinadas, letras incorrectas, etc.).

Descripción Visual:  
  
- Entidades externas se dibujan como rectángulos.  
- Procesos se representan con círculos o elipses.  
- Almacenes de datos se muestran como líneas dobles o cajas abiertas en uno de sus lados.  
- Las flechas indican el flujo de información entre las entidades, procesos y almacenes de datos.  
  
Ejemplo de Flujos:  
  
1. El jugador inicia el juego (flujo hacia el proceso "Iniciar Juego").  
2. El sistema selecciona una palabra (flujo del proceso "Seleccionar Palabra" hacia el almacén "Lista de Palabras").  
3. El sistema muestra la palabra oculta al jugador (flujo desde "Mostrar Palabra" hacia la entidad "Jugador").  
4. El jugador ingresa una letra (flujo desde "Jugador" hacia "Solicitar Letra").  
5. El sistema verifica la letra (flujos de entrada a "Verificar Letra" desde "Solicitar Letra" y desde "Lista de Palabras").  
6. El estado se actualiza, y se muestra si ganó o perdió.  
  
Recuerda que esta descripción te ayudará a crear tu propio DFD en papel o utilizando herramientas de diagramación. Si necesitas más ayuda o detalles específicos, ¡no dudes en preguntar!

Arquitectura de la Aplicación del Juego del Ahorcado



1. Capa de Presentación (Frontend)  
Esta capa es responsable de la interfaz de usuario y de cómo los jugadores interactúan con el juego.  
  
- Interfaz de Usuario:  
- Pantalla de inicio (botones para iniciar juego, instrucciones, etc.).  
- Pantalla de juego (visualización de la palabra oculta, letras adivinadas, intentos restantes).  
- Pantalla de resultados (mensaje de victoria o derrota, opción para reiniciar).  
  
- Tecnologías Sugeridas:  
- HTML/CSS para la estructura y estilo.  
- JavaScript para la interacción dinámica (por ejemplo, gestionar eventos de teclado).  
  
2. Capa de Lógica de Negocio (Backend)  
Esta capa gestiona la lógica del juego, los procesos y las interacciones entre el jugador y el juego.  
  
- Componentes Principales:  
- Gestor de Partidas: Controla el inicio de las partidas, las reglas del juego y la transición de estados.  
- Generador de Palabras: Selecciona aleatoriamente palabras de una lista predefinida.  
- Verificador de Letras: Comprueba si la letra ingresada por el jugador está en la palabra secreta.  
- Gestor de Estado: Mantiene el estado actual del juego, incluyendo la palabra oculta, letras adivinadas y intentos restantes.  
  
- Tecnologías Sugeridas:  
- Node.js, Python Flask o Ruby on Rails para implementar la lógica del servidor.  
- Bases de datos como MongoDB o SQLite para almacenar palabras y estadísticas de juegos.  
  
3. Capa de Almacenamiento (Base de Datos)  
Esta capa almacena toda la información necesaria para el funcionamiento del juego.  
  
- Componentes:  
- Lista de Palabras: Almacenamiento de palabras que se pueden seleccionar para las partidas.  
- Historial de Partidas: Registro de estadísticas de partidas anteriores (jugador, palabra, resultado, fecha).  
  
- Posibles Tecnologías:  
- Bases de datos relacionales (PostgreSQL, MySQL) o NoSQL (MongoDB).  
  
4. Capa de Comunicación  
Esta capa gestiona la comunicación entre el frontend y el backend.  
  
- API RESTful: Una interfaz de programación de aplicaciones (API) que permite al frontend enviar solicitudes al servidor y recibir respuestas. Los endpoints podrían incluir:  
- `POST /start-game`: Inicia una nueva partida.  
- `POST /guess-letter`: Envia la letra adivinada por el jugador.  
- `GET /results`: Obtiene los resultados de la partida.  
  
Flujo de Trabajo General  
1. El jugador inicia la aplicación desde su dispositivo.  
2. El jugador selecciona "Iniciar Juego", lo que envía una solicitud al backend para comenzar una nueva partida.  
3. El sistema selecciona una palabra y muestra la pantalla de juego al jugador, con la palabra oculta representada por guiones.  
4. El jugador ingresa letras, que se envían al backend para verificación.  
5. El sistema verifica la letra y actualiza el estado del juego (palabra revelada, intentos restantes).  
6. El flujo continúa hasta que el jugador gane o pierda, momento en el cual se muestran los resultados y la opción de reiniciar el juego.

**Programación del Juego del ahorcado (código)**

# Lista que contiene las representaciones en arte ASCII del ahorcado para cada intento fallido.

dibujo\_ahorcado = [

    '''

        +---+

        |   |

            |

            |

            |

            |

    =========

    ''',

    '''

        +---+

        |   |

        O   |

            |

            |

            |

    =========

    ''',

    '''

        +---+

        |   |

        O   |

        |   |

            |

            |

    =========

    ''',

    '''

        +---+

        |   |

        O   |

       /|   |

            |

            |

    =========

    ''',

    '''

        +---+

        |   |

        O   |

       /|\  |

            |

            |

    =========

    ''',

    '''

        +---+

        |   |

        O   |

       /|\  |

       /    |

            |

    =========

    ''',

]

# Palabra secreta que el jugador debe adivinar. Se convierte en una lista de caracteres.

palabra = list('calavera')

# Lista que representa la palabra con guiones bajos, lo que el jugador ve.

palabra\_oculta =['\_']\*len(palabra)

# Número de intentos que el jugador tiene.

intentos = 6

# Lista de todas las letras válidas del abecedario en español.

lista\_abecedario = list('abcdefghijklmnñopqrstuvwxyz')

# Lista para guardar las letras que el jugador ha descartado (ha fallado).

letras\_descartadas = []

# Función para mostrar el estado actual del juego.

def mostrar\_estado():

    # Muestra los intentos restantes, las letras falladas y el progreso de la palabra.

    print(f'Intentos restantes: {intentos}')

    print(f'Letras descartadas: {", ".join(letras\_descartadas)}\n')

    print(f'Palabra: {" ".join(palabra\_oculta)}\n')

    # Imprime el dibujo del ahorcado según el número de intentos restantes.

    print(dibujo\_ahorcado[6 - intentos])

# Función para verificar si la letra introducida por el usuario es válida.

def letra\_valida(letra):

    # Comprueba si el usuario introdujo más de una letra.

    if len(letra) != 1 :

        print('Has puesto más de una letra, inténtalo de nuevo.')

        return False

    # Comprueba si la letra está en el abecedario.

    elif letra not in lista\_abecedario:

        print('No has introducido una letra del abecedario')

        return False

    # Comprueba si la letra ya ha sido adivinada.

    elif letra in palabra\_oculta:

        print('La letra que has introducido ya la has acertado, inténtalo de nuevo.')

        return False

    # Comprueba si la letra ya ha sido descartada.

    elif letra in letras\_descartadas:

        print('Esa letra ya la habías dicho, inténtalo de nuevo.')

        return False

    # Si pasa todas las comprobaciones, la letra es válida.

    else:

        return True

# Función para gestionar una letra que el jugador ha adivinado correctamente.

def gestion\_letra(letra):

    # Itera sobre la palabra secreta para encontrar todas las ocurrencias de la letra.

    for i in range(len(palabra)):

        if palabra[i] == letra:

            # Reemplaza el guion bajo en la palabra oculta por la letra correcta.

            palabra\_oculta[i] = letra

            # Marca la letra en la palabra secreta como ya encontrada para evitar duplicados.

            palabra[i] = '\_'

# Inicio del juego: imprime un mensaje de bienvenida y las reglas.

print('BIENVENIDO AL JUEGO DEL AHORCADO 🎃 EDICIÓN HALLOWEEN 🎃\n')

print('Reglas del juego: Introduce letras para adivinar la palabra oculta .')

print(f'Tienes {intentos} intentos. ¡Buena suerte!')

# Bucle principal del juego: continúa mientras queden intentos y la palabra no esté completa.

while intentos > 0 and '\_' in palabra\_oculta:

    # Muestra el estado actual del juego.

    mostrar\_estado()

    print('-'\*50)

    # Pide al usuario que introduzca una letra y la convierte a minúscula.

    letra = input('INTRODUCE LETRA: ').lower()

    # Bucle para validar la entrada del usuario. Se repite hasta que se introduzca una letra válida.

    while not letra\_valida(letra):

        letra = input('INTRODUCE OTRA LETRA: ').lower()

    # Comprueba si la letra está en la palabra secreta.

    if letra in palabra:

        # Si acierta, llama a la función para actualizar la palabra oculta.

        gestion\_letra(letra)

        print('-'\*50)

        print('¡Has acertado la letra! Sigue así.')

    else:

        # Si falla, reduce un intento y añade la letra a la lista de descartadas.

        print('-'\*50)

        print('¡Has fallado la letra!')

        letras\_descartadas.append(letra)

        intentos -= 1