Name goes here

Title goes here

Nombre del estudiante: Laura Milena Noboa Zárate
Nombre de la asignatura: Lógica de la Programación
Fecha de elaboración del documento: 14 de enero del 2026

Diseño de Diagramas de Funcionalidad y Arquitectura Previo al Desarrollo de un Software

Índice

Introducción y Descripción del Software

El Software a ser desarrollado será: El juego del Ahorcado

El Juego del Ahorcado es un clásico juego de adivinanzas en el que el jugador intenta adivinar una palabra secreta letra por letra. El sistema elige una palabra al azar y el jugador tiene un número limitado de intentos (6) para adivinarla.

Por cada letra incorrecta, se dibuja una parte del ahorcado. El juego termina cuando el jugador adivina la palabra completa (victoria) o cuando se dibujan todas las partes del ahorcado (derrota)."

El proceso que sigue el software proceso es el siguiente:

1. El usuario visualiza el menú, para que el juego pueda desarrollarse debe iniciar partida
2. Al iniciar partida el software automáticamente elegirá una palabra al azar y la mostrar en la interfaz gráfica al mostrar la palabra en guiones en blanco y mostrara el numero de intentos que tiene disponible el usuario.
3. Al usurio haber intentado adivinar la palabra y haber fallado, el sistema automáticamente restará el número de fallos a su número de intentos, y proporcionará una lista de las letras usadas erróneamente, con el fin de no volver a utilizar la misma letra nuevamente.
4. Si el usuario adivina todas las letras de la palabra secreta antes de agotar su número de intentos ganará la partida, y si por el contrario, el jugador no adivina la palabra y agota su número de intentos, perderá la partida.
5. Al terminar la partida, el jugador se dirigirá al menú nuevamente, ahi tendra la opcion de jugar nuevamente o salir del juego.

¿Por qué lo elegí?

Elegí este programa porque presenta un buen balance entre simplicidad y complejidad. Permite aplicar conceptos de decisiones condicionales, bucles, manejo de datos (listas/arreglos) y organización de código en módulos.

Además, es un juego que casi todos conocen, lo que facilita entender los requisitos.

¿Cual es su objetivo?

El objetivo de este trabajo autónomo es documentar el diseño del Ahorcado mediante diagramas de funcionalidad y arquitectura, estableciendo las bases para una buena implementación posterior en cualquier lenguaje de programación.

Investigación Corta

Existen múltiples diagramas estandarizados para modelar la funcionalidad y arquitectura de aplicaciones, principalmente basados en UML y enfoques como C4 o DFD. Estos permiten detallar componentes, flujos, interacciones y despliegues con precisión técnica.

Diagramas de Arquitectura

Representan la estructura estática del sistema.

- Diagrama de Componentes (UML): Muestra módulos reutilizables, interfaces (providor/requeridor), puertos y dependencias; ideal para integraciones cliente-servidor o microservicios.
- Diagrama de Despliegue (UML): Ilustra nodos hardware (servidores, dispositivos), artefactos software distribuidos y conexiones físicas (redes, ejecución).
- C4 Model: Jerarquía escalable: Contexto (sistema + usuarios/externos), Contenedores (apps/servicios), Componentes (lógica interna) y Código (clases/detalles).

Diagramas de Funcionalidad

Visualizan procesos dinámicos y flujos de datos.

- Diagrama de Flujo de Datos (DFD): Identifica entidades externas, procesos (círculos), flujos (flechas) y almacenes (rectángulos abiertos); descompone en niveles para complejidad.
- Diagrama de Secuencia (UML): Líneas de vida de objetos, mensajes síncronos/asíncronos, activaciones y retornos; enfocado en orden temporal de interacciones.
- Diagrama de Actividad (UML): Nodos de acción, decisiones (rombos), bifurcaciones (barras), uniones y swimlanes para responsabilidades paralelas.

Diagramas Relacionales y Dinámicos

Tipo	Descripción Detallada	Uso Principal
ERD (Entidad-Relación)	Entidades (rectángulos), atributos (óvalos), relaciones (rombos) con cardinalidades (1:1, 1:N, N:M); soporta claves primarias/foráneas.	Diseño de bases de datos relacionales.
Diagrama de Colaboración (UML)	Objetos, mensajes, enlaces y frames de referencia; captura runtime para escenarios de casos de uso específicos.	Análisis dinámico de ejecuciones.
Diagrama de Paquetes (UML)	Paquetes con dependencias, imports/merges; organiza código en namespaces jerárquicos.	Estructura modular de código fuente.

Análisis del Problema (Software)

1. Identificar el problema.

El problema: “Diseñar un programa que permita jugar al Ahorcado entre un usuario y el computador, donde el sistema elige una palabra y el usuario intenta adivinarla con intentos limitados.”

2.Comprender el problema

Para poder desarrollar correctamente el programa hay que definir las incógnitas a resolver.

- Entradas: letras del usuario, opción de jugar de nuevo y opción de salir del juego.
- Salidas: palabra oculta, intentos, mensajes, dibujo del ahorcado.
- Reglas: no se pueden repetir las letras por esta razón implementamos la lista de palabras utilizadas equivocadamente, máximo 6 intentos, ganar si completa la palabra, perder si se acaban los intentos y el jugador ingresa una letra por vez.
- Usos: este programa puede ser usado por un solo jugador o por muchos jugadores ya sea formando equipos o por separado, también puede utilizarse para una competencia de memoria, razonamiento y velocidad al ir adivinando y completando la palabra.

3. Identificar soluciones alternativas

La principal razón por la que yo elijo esta alternativa es debido a poder manejar una priorización del software, cabe recalcar que esta solución esta enfocada en la formación de la estructura interna del software no en el desarrollo de las líneas de código.

Al programar tenemos distintos lenguajes a ser desarrollados, esto depende del nivel de conocimiento y manejo del mismo.

- **Solución 1: Programa de consola sin arquitectura (todo en una sola clase o archivo)**
  En esta opción, todo el código del juego del Ahorcado se escribe junto: lectura de letras, lógica del juego, impresión de mensajes y control de intentos en el mismo bloque de programa.

Ventaja: Es más rápida de escribir al inicio y puede ser suficiente para programas muy pequeños.
Desventaja: El código se vuelve difícil de entender y modificar, porque no hay separación entre datos, reglas del juego y la interacción con el usuario.

- **Solución 2: Aplicación de consola con arquitectura en capas**
  Aquí el juego sigue siendo de consola, pero se separa en tres partes generales:
  - Capa de presentación: se encarga de mostrar mensajes en pantalla y leer las letras que ingresa el jugador.
  - Capa de lógica/negocio: contiene las reglas del Ahorcado (validar letras, actualizar intentos, comprobar victoria o derrota, etc.).
  - Capa de datos: podría encargarse de cómo se obtiene la palabra secreta (lista fija, archivo, etc.).
    Ventaja: Hay más orden que en la solución 1 y es más fácil cambiar una parte sin romper las demás.
- **Solución 3 (ELEGIDA): Aplicación de consola con arquitectura MVC**
  En esta opción se mantiene la interfaz de consola, pero el código se organiza usando el patrón Modelo–Vista–Controlador (MVC):
  - Modelo: representa los datos y reglas del juego del Ahorcado (palabra secreta, letras acertadas, letras falladas, número de intentos, comprobación de estado del juego).
  - Vista: muestra por consola el estado del juego (palabra con guiones, intentos restantes, mensajes de victoria o derrota) y recibe las letras que ingresa el jugador.
  - Controlador: coordina el flujo del juego, pide datos a la Vista, llama al Modelo para validar la jugada, y decide cuándo la partida continúa o termina.
    Ventajas:
  - El código queda más organizado y cada parte tiene una responsabilidad clara.
  - Facilita futuros cambios, por ejemplo cambiar la Vista de consola a gráfica sin tocar la lógica del juego.

Justificación de la solución elegida

Para este proyecto se selecciona la **Solución 3: aplicación de consola con arquitectura MVC**, porque:

- Mantiene la simplicidad de una aplicación de consola, adecuada para el nivel del curso.
- Aplica un patrón de diseño ampliamente utilizado (MVC), que separa datos, presentación y control, facilitando el mantenimiento y la comprensión del programa.

4. Seleccionar la mejor solución

Para este proyecto se selecciona la alternativa: aplicación de consola con arquitectura MVC simplificada, porque es suficiente para cumplir el objetivo, es más fácil de implementar y permite separar bien la lógica del juego de la interfaz.

5. Listar los pasos de la solución

- [ ] Definir los elementos principales del juego
  - Decidir cómo se representará la palabra secreta, las letras acertadas, las letras falladas y el número máximo de intentos.
  - Definir qué mensajes verá el jugador en consola en cada momento (inicio, durante la partida, victoria, derrota).
- [ ] Diseñar el Modelo (M) del Ahorcado
  - Crear la estructura de datos para almacenar la palabra secreta, los intentos restantes, las letras usadas y el estado de victoria/derrota.
  - Definir operaciones del modelo: elegir palabra, comprobar letra, actualizar aciertos/fallos, comprobar si el juego terminó.
- [ ] Diseñar la Vista (V) en consola
  - Definir cómo se mostrará el tablero: palabra con guiones o letras descubiertas, número de intentos restantes y letras ya usadas.baulderasec.
  - Definir cómo se pedirán las entradas: mostrar instrucciones y leer la letra que el jugador ingresa por teclado.
- [ ] Diseñar el Controlador (C)
  - Diseñar el flujo principal del juego: iniciar partida, entrar en un bucle de turnos y terminar cuando gane o pierda.
  - Establecer cómo el Controlador usa el Modelo y la Vista: pide a la Vista que muestre el estado, obtiene la letra del usuario, llama al

Modelo para validar y decide si continúa o finaliza la partida.

- [ ] Definir el flujo detallado de una partida
  - Inicio:
    - Mostrar mensaje de bienvenida y reglas básicas.
    - El Modelo selecciona o carga la palabra secreta e inicializa los intentos.programaresdivertido.
  - Bucle de juego: se repite mientras queden intentos y no se haya adivinado toda la palabra.
    - La Vista muestra el estado actual (palabra parcial, intentos, letras usadas).
    - La Vista pide y lee una letra.
    - El Controlador envía la letra al Modelo para que la valide.
    - El Modelo actualiza aciertos/fallos e informa al Controlador si el juego sigue, se ganó o se perdió.
  - Fin:
    - Si ganó, la Vista muestra un mensaje de victoria y la palabra completa.
    - Si perdió, la Vista muestra la palabra secreta y un mensaje de derrota.baulderasec.
- [ ] Planificar posibles extensiones (opcional)
  - Permitir jugar varias partidas seguidas (el Controlador reinicia el Modelo).
  - Cambiar en el futuro la Vista de consola por una GUI o web manteniendo el mismo Modelo.

6. Evaluar / Probar la solución (del diseño)

En esta etapa se revisa si los diagramas de funcionalidad y arquitectura cubren todos los requisitos del problema del Ahorcado, identificando posibles fallos en el diseño antes de la implementación.

Casos de prueba del diseño:

- **Caso 1: Victoria** – Verificar si el diagrama de flujo contempla mostrar mensaje de victoria cuando todas las letras se adivinen antes de 6 intentos. Resultado: ✅ Cubierto en el flujo (bucle → palabra completa → fin victoria).

- **Caso 2: Derrota** – Confirmar que el flujo maneja agotar 6 intentos y mostrar la palabra secreta. Resultado: ✅ Cubierto (vidas=0 → fin derrota).
- **Caso 3: Letras repetidas** – Revisar si los casos de uso y flujo evitan restar intentos por letras ya usadas. Resultado: ✅ Cubierto en "Visualizar letras usadas" y validación en flujo.
- **Caso 4: Múltiples partidas** – Verificar que el menú permita regresar y reiniciar (nueva palabra, vidas). Resultado: ✅ Cubierto en "Regresar al menú" y MVC reinicia Modelo.
- **Caso 5: Entradas inválidas** – Confirmar que el diseño contemple validar solo letras. Resultado: ⚠️ Falta detalle explícito → Agregar validación en Controlador/Vista.
- **Criterios de éxito del diseño:**

▢ Todos los casos de uso del diagrama púrpura tienen flujo correspondiente en el diagrama vertical.
▢ La arquitectura MVC asigna correctamente las responsabilidades (Modelo: lógica; Vista: mostrar/leer; Controlador: coordinar).
▢ No hay contradicciones entre diagramas ni requisitos no cubiertos.

Referencias

Lucid Software. (2025, 8 octubre). *Cómo hacer 5 tipos de diagramas de arquitectura de software.* Lucidchart. https://www.lucidchart.com/blog/es/como-elaborar-diagramas-de-arquitectura

Ckittel. (s. f.). *Creación de diagramas de diseño de arquitectura* - Microsoft Azure Well-Architected Framework. Microsoft Learn. https://learn.microsoft.com/es-es/azure/well-architected/architect-role/design-diagrams

Suárez, J. M. S. (2023, 6 mayo). *Diagramas de arquitectura con C4 model*. Adictos Al Trabajo. https://adictosaltrabajo.com/2023/05/06/diagramas-de-arquitectura-con-c4-model/

Vive UNIR. (2024, April 26). *Tipos de diagramas: cuáles existen y sus principales usos.* Universidad Virtual. | UNIR Ecuador - Maestrías Y Grados Virtuales; UNIR. https://ecuador.unir.net/actualidad-unir/tipos-diagramas/