Cristian Suárez

Lógica de Programación

Universidad Internacional

1. Diagramas de Funcionalidad

Estos diagramas se centran en el comportamiento del sistema y en cómo los usuarios interactúan con él. Su objetivo es validar los requisitos.

Tipos principales:

Diagrama de Casos de Uso (UML): Muestra los actores (usuarios) y las operaciones (casos de uso) que realizan en el sistema. Es el nivel más alto de funcionalidad.

Diagrama de Actividad (UML): Representa flujos de trabajo paso a paso de las actividades y acciones. Es similar a un diagrama de flujo avanzado.

Diagrama de Secuencia (UML): Muestra cómo los objetos interactúan entre sí en un orden temporal específico para completar una función.

Diagrama de Flujo de Datos (DFD): Muestra cómo se mueve la información a través de un sistema (entradas, procesos y salidas).

Diagrama de Casos de Uso

2. Diagramas de Arquitectura de Aplicaciones

Estos diagramas muestran la estructura del software, sus módulos, tecnologías y cómo se comunican entre sí. Su objetivo es guiar el desarrollo técnico.

Tipos principales:

Diagrama de Componentes (UML): Desglosa el sistema en bloques lógicos (componentes) y muestra las dependencias entre ellos (ej. una API, una base de datos, un módulo de autenticación).

Diagrama de Despliegue (UML): Se centra en el hardware (nodos) y cómo el software se instala en ellos (servidores, nube, dispositivos).

Diagrama de Capas (Layered Pattern): Una vista lógica que organiza el código en capas horizontales (Presentación, Negocio, Datos).

Modelo C4 (Contexto, Contenedores, Componentes, Código): Un enfoque moderno que permite hacer "zoom" desde una vista general hasta el detalle del código.

Diagrama de Componentes

Pasos para desarrollar el Juego Piedra, papel o tijera.

Paso 1: Definición del Problema (El "Qué")

Debemos definir claramente qué queremos que haga el sistema para evitar ambigüedades.

Objetivo: Crear un sistema que permita un juego de azar simple entre un usuario humano y la computadora (IA).

Condición de Victoria: El sistema debe decidir un ganador basándose en reglas preestablecidas de jerarquía circular.

Alcance (Scope):

Es un juego de 1 contra 1 (Usuario vs. CPU).

Es una sola ronda por ejecución (o un bucle hasta que el usuario decida salir).

No habrá apuestas ni multijugador en red por ahora.

Paso 2: Análisis de Datos (Modelo E-P-S)

Aquí identificamos qué información entra, cómo se transforma y qué sale. Esto es vital para diseñar luego las variables.

A. Entradas (Inputs) - ¿Qué necesita el sistema?

Elección del Usuario: Un valor que represente Piedra, Papel o Tijera.

Reto a resolver: ¿Cómo introduce esto el usuario? (¿Texto "Piedra"? ¿Un número 1, 2, 3? ¿Un clic?). Debemos normalizar esta entrada.

Elección del Sistema (CPU):

Reto a resolver: La computadora no "piensa". Necesitamos generar un valor aleatorio que represente una de las tres opciones con igual probabilidad (33.3%).

B. Procesos (Logica Interna) - ¿Qué hace el sistema?

Validación: Asegurar que lo que ingresó el usuario sea válido (ej: que no escriba "Fuego").

Generación Aleatoria: Ejecutar la función de azar para la CPU.

Comparación (El núcleo del problema): Confrontar la Elección_Usuario vs Elección_CPU.

Determinación de Estado: Asignar el resultado: Ganar, Perder o Empatar.

C. Salidas (Outputs) - ¿Qué ve el usuario?

Mostrar qué eligió la CPU (para dar transparencia al juego).

Mostrar el mensaje del resultado final ("¡Ganaste!", "Empate", etc.).

Paso 3: Las Reglas del Negocio (Matriz de Lógica)

En programación, no podemos decirle a la computadora "ya sabes cómo se juega". Debemos ser explícitos. Aquí definimos la lógica de comparación que luego se convertirá en estructuras condicionales (if/else).

Paso 4: Identificación de Restricciones y Casos Borde

Un buen análisis de problemas anticipa dónde puede fallar el software.

Entradas Inválidas (Dirty Data):

Problema: El usuario escribe "Roca" en lugar de "Piedra" o escribe "Piedra " (con espacio).

Solución Lógica: El sistema debe limpiar la entrada (quitar espacios, convertir a minúsculas) o rechazarla antes de jugar.

Equidad del Azar:

Problema: Si el generador aleatorio no es bueno, la CPU podría sacar siempre "Piedra".

Requisito: Asegurar que el algoritmo de aleatoriedad tenga una distribución uniforme.

Diseño de Funcionalidades

1. Identificación de Actores

Antes de dibujar, definimos quién interactúa con el sistema.

Actor Principal: Jugador (Usuario) - Quien inicia la acción.

Actor Secundario/Sistema: CPU (IA Simple) - El componente que responde y genera la jugada aleatoria.

2. Diseño del Diagrama de Casos de Uso (UML)

Este diagrama define las "burbujas" de funcionalidad que tendrá tu software.

Descripción visual del diseño: Imagina una caja rectangular que representa los límites de tu sistema ("La Aplicación"). Fuera de ella está el "Jugador". Dentro, tenemos las siguientes funciones (Casos de Uso):

Iniciar Partida: El disparador del juego.

Seleccionar Movimiento: La acción principal del jugador (elegir Piedra, Papel o Tijera).

Validar Selección: Una función interna (marcada con <<include>>) que se asegura de que el usuario no ingresó basura.

Generar Movimiento CPU: Función interna donde el sistema actúa.

Calcular Ganador: El motor lógico que compara las dos elecciones.

Mostrar Resultado: La interfaz final que comunica quién ganó.

3. Especificación Detallada de los Casos de Uso

Un diagrama son solo dibujos si no tiene una narrativa. Aquí diseñamos el "contrato" de comportamiento para la funcionalidad principal: "Jugar una Ronda".

ID Caso de UsoCU-01: Jugar RondaActorJugadorPrecondiciónEl programa debe estar iniciado y listo para recibir input.Flujo Principal (Camino Feliz)

1. El Sistema solicita al Jugador que elija una opción.

2. El Jugador ingresa su elección (Piedra, Papel, Tijera).

3. El Sistema Valida la entrada (ver Flujo Alternativo A).

4. El Sistema ejecuta Generar Movimiento CPU (azar).

5. El Sistema Calcula el Ganador comparando ambas elecciones.

6. El Sistema muestra: Elección CPU + Mensaje de Victoria/Derrota.

Flujo Alternativo A (Error)

3a. El Jugador ingresa un texto inválido (ej: "Fuego").

3b. El Sistema muestra mensaje: "Opción no válida".

3c. El Sistema vuelve al paso 1.

juego termina la ronda y pregunta si se desea jugar otra vez.

4. Diagrama de Actividad (Flujo Funcional)

Mientras el diagrama de Casos de Uso muestra qué hace el sistema, el diagrama de actividad muestra el orden lógico de esas funciones. Este diseño es vital para entender los condicionales (rombos de decisión).

Diseño del Flujo:

Inicio (Círculo negro).

Acción: Solicitar Entrada.

Decisión (Rombo): ¿Entrada válida?

No: Volver a solicitar.

Sí: Continuar.

Acción Paralela:

Guardar variable JUGADOR.

Generar variable CPU (Random).

Decisión Múltiple (Lógica del Juego):

¿Son iguales? -> Empate.

¿Gana Jugador? -> Victoria.

¿Gana CPU? -> Derrota.

Acción: Imprimir Resultado.

Fin (Círculo doble).

Diseño del Diagrama de Componentes

Imagina el sistema como un edificio de tres pisos. La información fluye de arriba hacia abajo y las respuestas de abajo hacia arriba.

1. Capa de Presentación (Interfaz de Usuario / UI)

Es la "cara" del software. Es lo único que el usuario ve.

Componente: GestorDeEntradaSalida (Input/Output Handler).

Responsabilidad:

Capturar lo que escribe el usuario (inputs).

Limpiar la pantalla o dar formato al texto.

Mostrar los mensajes finales ("Ganaste", "Perdiste").

Regla de Oro: Este componente no sabe jugar, solo sabe preguntar y mostrar.

2. Capa de Control (Lógica del Juego)

Es el "cerebro" o el director de orquesta.

Componente: ControladorDelJuego (Game Controller).

Responsabilidad:

Coordina la partida. Llama a la UI para pedir datos y luego llama al Motor para ver quién ganó.

Mantiene el estado de la sesión (ej. ¿Seguimos jugando o salimos?).

Maneja el bucle principal (while playing).

3. Capa de Núcleo (Core / Servicios)

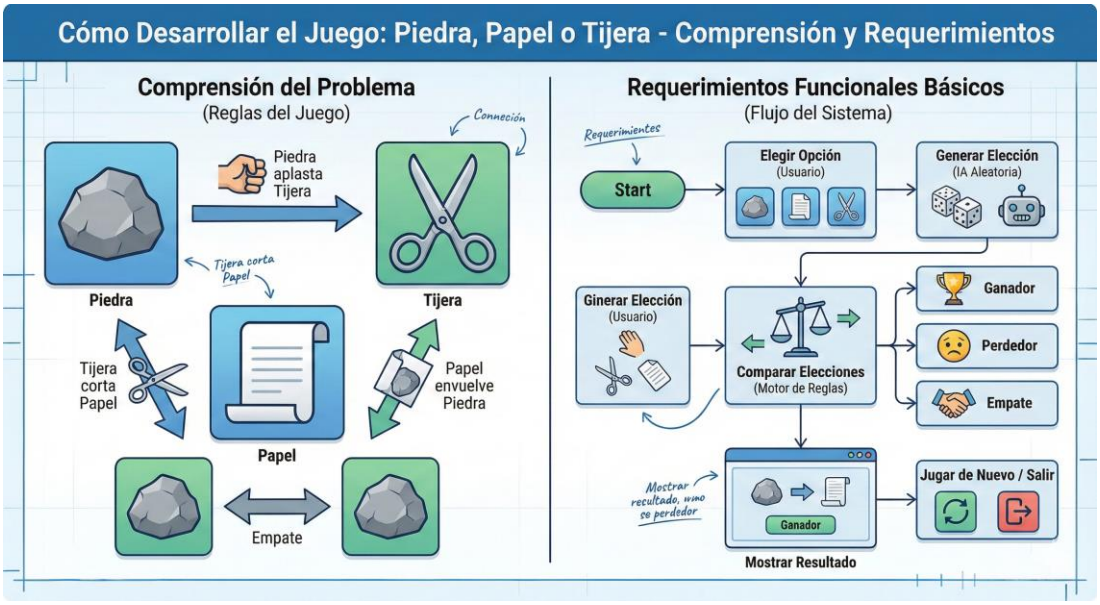
Son los expertos. No saben de colores ni de textos bonitos, solo saben de lógica pura y matemática.

Componente A: MotorDeReglas (Rules Engine).

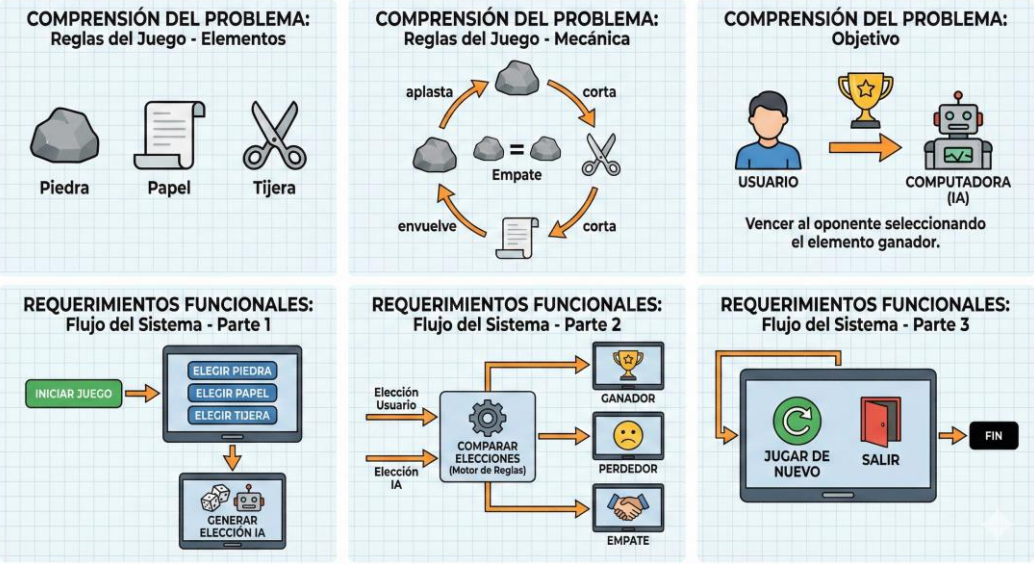
Responsabilidad: Contiene la matriz de verdad. Recibe dos valores (ej: "Piedra", "Papel") y devuelve el resultado ("Gana Papel").

Componente B: GeneradorAleatorio (RNG Service).

Responsabilidad: Genera la elección de la CPU de forma justa y equiprobable.


Cómo Desarrollar el Juego: Piedra, Papel o Tijera - Comprensión y Requerimientos
Comprensión del Problema
(Reglas del Juego)
Piedra aplasta Tijera
Piedra
Tijera
Tijera corta Papel
Papel envuelve Piedra
Papel
Empate
Requerimientos Funcionales Básicos
(Flujo del Sistema)
Start
Elegir Opción
(Usuario)
Generar Elección
(IA Aleatoria)
Ginerar Elección
(Usuario)
Comparar Elecciones
(Motor de Reglas)
Ganador
Perdedor
Empate
Ganador
Mostrar Resultado
Jugar de Nuevo / Salir

+---


COMPRENSIÓN DEL PROBLEMA:
Reglas del Juego - Elementos
Piedra
Papel
Tijera
COMPRENSIÓN DEL PROBLEMA:
Reglas del Juego - Mecánica
aplasta
corta
Empate
envuelve
corta
COMPRENSIÓN DEL PROBLEMA:
Objetivo
USUARIO
COMPUTADORA (IA)
Vencer al oponente seleccionando el elemento ganador.
REQUERIMIENTOS FUNCIONALES:
Flujo del Sistema - Parte 1
INICIAR JUEGO
ELEGIR PIEDRA
ELEGIR PAPEL
ELEGIR TIJERA
GENERAR ELECCIÓN IA
REQUERIMIENTOS FUNCIONALES:
Flujo del Sistema - Parte 2
Elección Usuario
Elección IA
COMPARAR ELECCIONES (Motor de Reglas)
GANADOR
PERDEDOR
EMPATE
REQUERIMIENTOS FUNCIONALES:
Flujo del Sistema - Parte 3
JUGAR DE NUEVO
SALIR
FIN