UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

Academia de Informática Cuautepec
Ingeniería de Software
Construcción y Evolución de Software
Profesor Emmanuel Vite Sevilla

Nombre de la práctica: Programa uno

Número: 1

Alumno(s)/Equipo:

Axel Javier Rios Sánchez

Jesús Josué Suarez López

Evelin Maldonado Espinosa

Mauricio Charbel Chavez Galindo

Fecha:27/02/2026

Objetivo: Desarrollar un programa que simule el funcionamiento del famoso juego UNO, aplicando sus reglas básicas, turnos con el fin de poner en práctica los conocimientos adquiridos en las materias anteriores de la carrera de Ingeniería de Software.

I.- INTRODUCCIÓN

El juego UNO es uno de los juegos de cartas más populares a nivel mundial, caracterizado por su dinamismo y reglas sencillas basadas en la coincidencia de color o número. Su estructura por turnos lo convierte en un excelente ejemplo práctico para el desarrollo de programas que simulen situaciones reales mediante lógica y control de flujo.

En la presente práctica se desarrollará un programa que simule el funcionamiento básico del juego UNO, respetando sus reglas principales como la coincidencia por color o número y la dinámica de turnos entre jugadores. Por ahora, no se implementarán las cartas especiales, con el fin de enfocarse primero en la estructura fundamental del juego y en la correcta aplicación de la lógica de programación.

A través de este proyecto se pondrán en práctica conceptos fundamentales de programación y principios básicos de la Ingeniería de Software, tales como el análisis del problema, el diseño de la solución y la organización estructurada del código.

II.- ANTECEDENTES (MARCO TÉORICO)

Construcción de software:

La construcción de software es una fase fundamental dentro de la Ingeniería de Software, ya que en ella se lleva a cabo la implementación del sistema previamente analizado y diseñado. En esta etapa se traduce la solución planteada en código fuente utilizando un lenguaje de programación, aplicando buenas prácticas de estructuración, organización y claridad.

Para el desarrollo del Programa Uno, la construcción de software implica la implementación de la lógica que permite simular el funcionamiento básico del juego UNO. Esto incluye la definición de estructuras de datos para representar el mazo y las cartas, el uso de estructuras de control para validar jugadas y el manejo adecuado de los turnos entre jugadores.

Durante esta fase se aplican conceptos como el uso de variables, listas, condicionales y ciclos, los cuales permiten modelar el comportamiento del juego de manera ordenada. La construcción correcta del software garantiza que el programa funcione conforme a los requisitos establecidos y facilita futuras mejoras, como la incorporación de cartas especiales.

III.- HARDWARE / SOFTWARE NECESARIO Hardware:

Hardware:

- Computadora o laptop con procesador básico (Intel i3 o equivalente en adelante).
- Memoria RAM mínima de 4 GB.
- Teclado y mouse.
- Monitor para la visualización del programa.

Software:

- Sistema operativo (Windows, Linux o macOS).
- Entorno de desarrollo o editor de código: Java online
- Lenguaje de programación a utilizar Java

IV.- DESARROLLO

Las reglas básicas del uno

1. Preparación y Turnos

- Se reparten 7 cartas a cada jugador.
- Se voltea una carta del mazo para iniciar la pila de descartes.
- En tu turno, debes poner una carta que coincida con la de la pila por color, número o símbolo.
- Si no tienes una carta válida, debes robar una del mazo. Si esa carta te sirve, puedes jugarla de inmediato; si no, tu turno termina.

2. Cartas Especiales y Comodines

- +2 (Toma dos): El siguiente jugador roba 2 cartas y pierde su turno.
- Reversa: Cambia el sentido del juego (de derecha a izquierda o viceversa).
- Salto: El siguiente jugador pierde su turno.

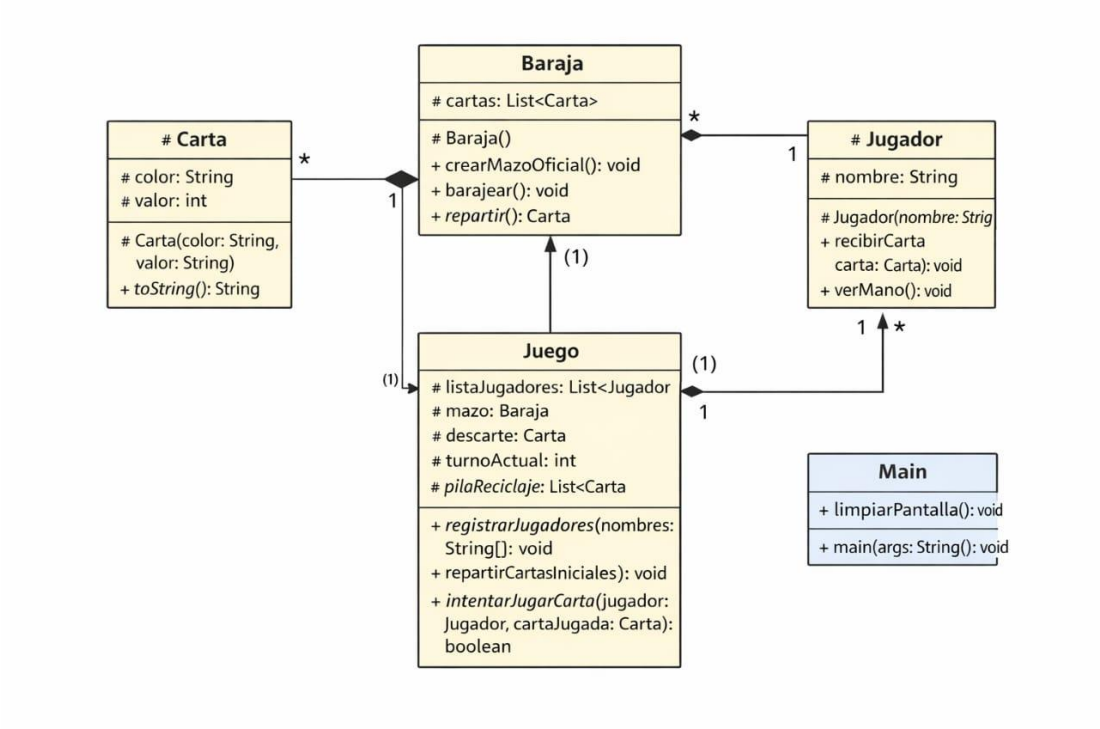
- Comodín Color: Cambia el color del juego a tu elección.
- Comodín +4 (Toma cuatro): El siguiente jugador roba 4 cartas, pierde su turno y tú eliges el color. Solo debe usarse si no tienes cartas del color actual en tu mano.

3. Reglas Críticas (Que muchos olvidan)

- ¡Gritar "UNO"!: Debes gritarlo justo antes de poner tu penúltima carta. Si otro jugador te atrapa con una sola carta sin haberlo dicho, debes robar 2 cartas de penalización.
- No se "acumulan" los +2: Según las reglas oficiales de Mattel, si alguien tira un +2, el siguiente debe robar y no puede tirar otro +2 encima para que el tercero robe 4.
- Desafío del +4: Si sospechas que alguien tiró un +4 teniendo el color que estaba en juego, puedes desafiarlo. Si es culpable, roba 4; si es inocente, tú robas

Para desarrollar el programa establecido es necesario definir: el diagrama UML y el código construido

Diagrama UML



import java.util.*;

```
class Carta {
    String color;
    String valor;

    public Carta(String color, String valor) {
        this.color = color;
        this.valor = valor;
    }

    @Override
    public String toString() {
        return  "\n+----------+" +
                "\n| " + String.format("%-2s", valor) + "        |" +
                "\n|          |" +
                "\n|" + String.format("%-10s", color) + "|" +
                "\n|          |" +
                "\n|       " + String.format("%2s", valor) + " |" +
                "\n+----------+";
    }
}

class Baraja {
    ArrayList<Carta> cartas = new ArrayList<>();

    public Baraja() {
        crearMazoOficial();
```

```
        barajear();
    }

    public void crearMazoOficial() {
        String[] colores = {"Rojo", "Amarillo", "Verde", "Azul"};

        for (String color : colores) {
            cartas.add(new Carta(color, "0"));
            for (int i = 1; i <= 9; i++) {
                cartas.add(new Carta(color, String.valueOf(i)));
                cartas.add(new Carta(color, String.valueOf(i)));
            }
        }
    }

    public void barajear() {
        Collections.shuffle(cartas);
    }

    public Carta repartir() {
        if (cartas.size() > 0) {
            return cartas.remove(cartas.size() - 1);
        }
        return null;
    }
}
```

```
class Jugador {
    String nombre;
    ArrayList<Carta> mano = new ArrayList<>();

    public Jugador(String nombre) {
        this.nombre = nombre;
    }

    public void recibirCarta(Carta carta) {
        if (carta != null) {
            mano.add(carta);
        }
    }

    public void verMano() {
        System.out.println("Mano de " + nombre + ":");
        for (int i = 0; i < mano.size(); i++) {
            System.out.println("[" + i + "] " + mano.get(i));
        }
    }
}

class Juego {
    ArrayList<Jugador> listaJugadores = new ArrayList<>();
    Baraja mazo = new Baraja();
    Carta descarte;
    int turnoActual = 0;
```

```
ArrayList<Carta> pilaReciclaje = new ArrayList<>();

public void registrarJugadores(String[] nombres) {
    for (String nombre : nombres) {
        listaJugadores.add(new Jugador(nombre));
    }
}

public void repartirCartasIniciales() {
    for (Jugador jugador : listaJugadores) {
        for (int i = 0; i < 7; i++) {
            jugador.recibirCarta(mazo.repartir());
        }
    }
}

public boolean intentarJugarCarta(Jugador jugador, Carta cartaJugada) {
    if (descarte == null ||
        cartaJugada.color.equals(descarte.color) ||
        cartaJugada.valor.equals(descarte.valor)) {

        if (descarte != null) {
            pilaReciclaje.add(descarte);
        }

        descarte = cartaJugada;
        jugador.mano.remove(cartaJugada);
```

```
            System.out.println("Jugada exitosa");
            return true;

        } else {
            return false;
        }
    }
}

public class Main {

    public static void limpiarPantalla() {
        for (int i = 0; i < 40; i++) {
            System.out.println();
        }
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Introduce tu nombre: ");
        String nombreUsuario = sc.nextLine();

        Juego miJuego = new Juego();
        miJuego.registrarJugadores(new String[]{nombreUsuario, "CPU"});
        miJuego.repartirCartasIniciales();
```

```
miJuego.descarte = miJuego.mazo.repartir();

boolean juegoEnProgreso = true;
Random random = new Random();

while (juegoEnProgreso) {

    limpiarPantalla();

    if (miJuego.mazo.cartas.size() < 2) {
        miJuego.mazo.cartas.addAll(miJuego.pilaReciclaje);
        miJuego.pilaReciclaje.clear();
        miJuego.mazo.barajear();
        System.out.println("¡El mazo se había agotado! Reciclando cartas...");
    }

    System.out.println("\n--- Carta en la mesa: " + miJuego.descarte);
    Jugador jugadorActual = miJuego.listaJugadores.get(miJuego.turnoActual);

    // ===== TURNO DEL JUGADOR =====
    if (jugadorActual.nombre.equals(nombreUsuario)) {

        jugadorActual.verMano();
        boolean decidido = false;

        while (!decidido) {
```

```
                System.out.println("\n1. Poner una carta");
                System.out.println("2. Robar una carta");
                System.out.print("Selecciona opción: ");
                String opcion = sc.nextLine();

                if (opcion.equals("1")) {

                    try {
                        System.out.print("Elige índice: ");
                        int indice = Integer.parseInt(sc.nextLine());
                        Carta cartaSeleccionada = jugadorActual.mano.get(indice);

                        if (miJuego.intentarJugarCarta(jugadorActual,
cartaSeleccionada)) {

                            if (jugadorActual.mano.size() == 1) {
                                System.out.print("¡Te queda una carta! ¿Gritar UNO? (s): ");
                                String grito = sc.nextLine().toLowerCase();

                                if (!grito.equals("s")) {
                                    System.out.println("¡No gritaste UNO! +2 cartas.");
                                    jugadorActual.recibirCarta(miJuego.mazo.repartir());
                                    jugadorActual.recibirCarta(miJuego.mazo.repartir());
                                } else {
                                    System.out.println("¡Gritaste UNO!");
                                }
                            }
```

```
                        decidido = true;
                    } else {
                        System.out.println("Carta no válida.");
                    }

                } catch (Exception e) {
                    System.out.println("Índice inválido.");
                }

            } else if (opcion.equals("2")) {

                Carta cartaNueva = miJuego.mazo.repartir();
                System.out.println("Robaste: " + cartaNueva.color + " " +
cartaNueva.valor);

                if (!miJuego.intentarJugarCarta(jugadorActual, cartaNueva)) {
                    jugadorActual.recibirCarta(cartaNueva);
                }

                decidido = true;
            }
        }

    }
    // ===== TURNO DE LA CPU =====
    else {
```

```
            System.out.println("\nTurno de la CPU");
            boolean jugadaRealizada = false;

            for (Carta carta : jugadorActual.mano) {

                if (miJuego.intentarJugarCarta(jugadorActual, carta)) {

                    System.out.println("CPU jugó: " + carta.color + " " + carta.valor);

                    if (jugadorActual.mano.size() == 1) {

                        if (random.nextDouble() < 0.5) {
                            System.out.println("¡CPU grita: UNO!");
                        } else {
                            System.out.println("¡CPU olvidó gritar UNO! Penalización +2
cartas.");
                            jugadorActual.recibirCarta(miJuego.mazo.repartir());
                            jugadorActual.recibirCarta(miJuego.mazo.repartir());
                        }
                    }

                    jugadaRealizada = true;
                    break;
                }
            }
```

```
            if (!jugadaRealizada) {

                Carta cartaNueva = miJuego.mazo.repartir();

                if (miJuego.intentarJugarCarta(jugadorActual, cartaNueva)) {

                    System.out.println("CPU robó y jugó: " + cartaNueva.color + " " +
cartaNueva.valor);

                    if (jugadorActual.mano.size() == 1) {

                        if (random.nextDouble() < 0.7) {
                            System.out.println("¡CPU grita: UNO!");
                        } else {
                            System.out.println("¡CPU olvidó gritar UNO! Penalización +2
cartas.");
                            jugadorActual.recibirCarta(miJuego.mazo.repartir());
                            jugadorActual.recibirCarta(miJuego.mazo.repartir());
                        }
                    }

                } else {
                    jugadorActual.recibirCarta(cartaNueva);
                    System.out.println("CPU robó pero no pudo jugar.");
                }
            }
        }
```

```
            System.out.println("\nPresiona Enter para continuar...");
            sc.nextLine();

            if (jugadorActual.mano.size() == 0) {
                juegoEnProgreso = false;
                System.out.println("¡El jugador " + jugadorActual.nombre + " ha
ganado!");
            } else {
                miJuego.turnoActual =
                        (miJuego.turnoActual + 1) % miJuego.listaJugadores.size();
            }
        }

        sc.close();
    }
}
```

INFORME DE DOCUMENTACIÓN Y PRUEBAS (QA) - PROYECTO UNO

Versión: 1.1 | **Fecha:** 22 de febrero de 2026 **responsable de Calidad:** Evelin Maldonado

1. OBJETIVO DE LA DOCUMENTACIÓN

Este documento detalla el análisis de calidad realizado sobre la base funcional del juego UNO. Se centra en la validación de las reglas de negocio, la robustez del código frente a errores del usuario y la identificación de riesgos técnicos para las próximas semanas de desarrollo.

2. ANÁLISIS DE LA ESTRUCTURA (MAPA DE CLASES)

Se ha verificado que la arquitectura actual cumpla con los requisitos mínimos:

- **Clase Carta:** Define las propiedades básicas como Color y Valor. Se validó que el método de centrado visual funcione correctamente en consola.

- **Clase Baraja:** Gestiona el ciclo de vida del mazo de 108 cartas. Se confirmó que la lógica de barajado (Collections.shuffle) sea aleatoria.
- **Clase Juego:** Funciona como el motor de reglas. La separación del método esJugadaValida permite realizar pruebas sin necesidad de ejecutar todo el juego.
- **Clase Jugador:** Administra la mano de cartas. Se validó que el método recibirCarta gestione correctamente los objetos.

3. MATRIZ DE CASOS DE PRUEBA (TEST CASES)

Se ejecutaron los siguientes escenarios para garantizar la estabilidad del programa:

ID	Escenario de Prueba	Entrada	Resultado Esperado	Estado
TC-01	Coincidencia de Color	Mesa: Rojo / Mano: Rojo	Jugada permitida	**EXITOSO**
TC-02	Coincidencia de Valor	Mesa: Azul 7 / Mano: Verde 7	Jugada permitida	**EXITOSO**
TC-03	Jugada Inválida	Mesa: Amarillo 1 / Mano: Verde 8	Mostrar mensaje de error	**EXITOSO**
TC-04	Error de Tipo (Input)	Ingresar "Letras" en menú	Atrapado por Exception	**EXITOSO**
TC-05	Índice fuera de rango	Elegir Carta #50	Mostrar "No existe"	**EXITOSO**
TC-06	Mazo Agotado	Cartas mazo = 0	Reciclar pila de descarte	**EXITOSO**

4. ANEXO DE EVIDENCIAS TÉCNICAS (CAPTURAS)

TC-01,TC-02,TC-03:

El sistema valida la entrada del jugador comparando la carta elegida con la carta en mesa. Si las propiedades de Color o Valor no coinciden, el motor de reglas bloquea la acción y emite una alerta, garantizando que se respeten las normas básicas del juego documentadas en la matriz.

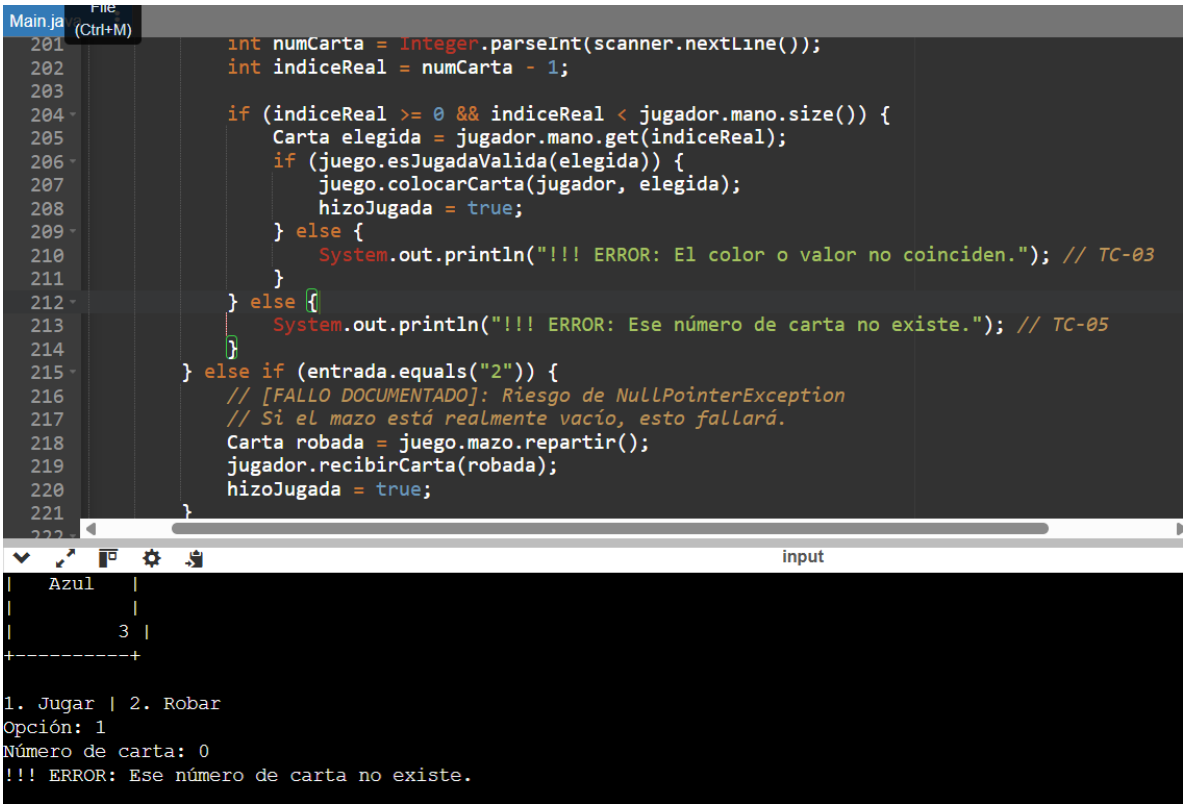


Foto 1: Evidencia Lógica Validación

Validación de Resiliencia (TC-06)

Se forzó el vaciado del mazo para comprobar el reciclaje de cartas.

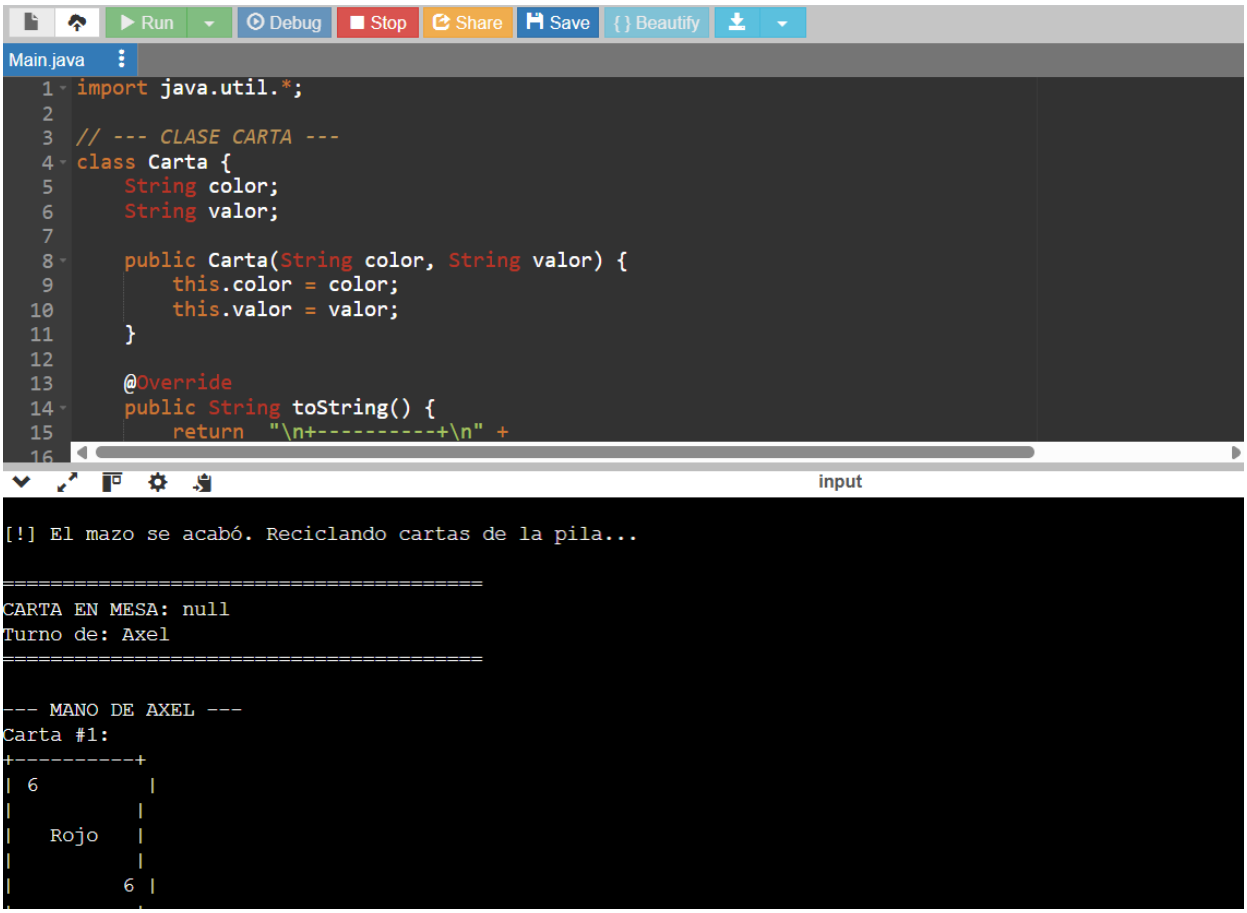


Foto 2: Mensaje de reciclaje de mazo tras agotarse las cartas.

Pruebas de Robustez (TC-04 y TC-05)

Validación del manejo de errores mediante bloques try-catch.

```
        } else if (entrada.equals("2")) {
            // [FALLO DOCUMENTADO]: Riesgo de NullPointerException
            // Si el mazo está realmente vacío, esto fallará.
            Carta robada = juego.mazo.repartir();
            jugador.recibirCarta(robada);
            hizoJugada = true;
        }
    } catch (NumberFormatException e) {
        System.out.println("!!! Por favor, ingresa solo números."); // TC-04
    }
}


vate static void turnoCPU(Juego juego, Jugador cpu) {
 // [FALLO DOCUMENTADO]: IA Lineal. La CPU juega lo primero que ve.
 Carta elegida = null;
 for (Carta c : cpu.mano) {
     if (juego.esJugadaValida(c)) {
         elegida = c;
         break;
     }
 }
 if (elegida != null) {
     juego.colocarCarta(cpu, elegida);
     System.out.println("La CPU jugó: " + elegida);
 } else {
     cpu.recibirCarta(juego.mazo.repartir());
     System.out.println("La CPU robó carta.");
 }
```

Foto 3: Implementación de seguridad para evitar cierres inesperados por entradas inválidas.

Identificación de Fallos de Lógica (Deuda Técnica)

Documentación de la ausencia de reglas críticas.

```
    System.out.println("======================================");

    if (jugadorActual.nombre.equals("Axel")) {
        // [FALLO DOCUMENTADO]: Aquí puedes capturar el TC-04 (letras) y TC-05 (índice 99)
        turnoHumano(miJuego, jugadorActual, scanner);
    } else {
        turnoCPU(miJuego, jugadorActual);
    }

    // [FALLO DOCUMENTADO]: Regla del "UNO" ausente
    // El juego termina sin preguntar si gritó UNO. Captura esto cuando ganes.
    if (jugadorActual.mano.isEmpty()) {
        System.out.println("\n¡EL JUGADOR " + jugadorActual.nombre.toUpperCase() + " HA GANADO
        juegoEnProgreso = false;
    } else {
        miJuego.turnoActual = (miJuego.turnoActual + 1) % miJuego.listaJugadores.size();
    }
```

Foto 4: El sistema permite ganar sin validar el grito de "UNO".

5. HALLAZGOS Y DEUDA TÉCNICA (QA FINDINGS)

Tras las pruebas, se identifican puntos que requieren atención para la Versión 2:

- **Regla del "UNO":** El sistema no detecta cuando a un jugador le queda una sola carta. Se recomienda añadir una validación de "Grito de UNO".
- **Cartas de Acción:** El motor de juego actual no procesa saltos de turno ni cambios de sentido.
- **Seguridad en el Robado:** Se debe validar qué sucede si el mazo y la pila están vacíos para evitar un NullPointerException.
- **IA de la CPU:** La lógica de la CPU es lineal. Se sugiere una mejora para que priorice cartas por color predominante.

6. CONCLUSIÓN DE CALIDAD

El software en su versión 1.1 se encuentra en estado **ESTABLE**. Las correcciones en el manejo de entradas han eliminado los cierres inesperados (crashes), cumpliendo satisfactoriamente con los entregables.