

# Documentación Proyecto Sudoku

## 1. Análisis de requerimientos

### Requisitos funcionales

- RF-001: Generación de tablero
- RF-002: Lógica interna de backtracking
- RF-003: Validación de movimientos
- RF-004: Colocación de números
- RF-005: Comprobación de final
- RF-006: Interfaz de consola (JuegoSudoku)
- RF-007: Interfaz gráfica Swing (SudokuGUI)
- RF-008: Reinicio y cambio de dificultad (GUI)

ID	Descripción	Prioridad	Fuente	Estado
RF-01	Generación de tablero 9x9 de Sudoku.	Alta	Requisitos internos	Completado
RF-02	Lógica interna de backtracking.	Alta	Requisitos internos	Completado
RF-03	Validación de movimientos.	Alta	Requisitos internos	Completado
RF-04	Colocación de números en casillas vacías	Alta	Requisitos internos	Completado
RF-05	Comprobación de que se ha resuelto correctamente el tablero	Alta	Requisitos internos	Completado
RF-06	Interfaz de consola con la clase JuegoSudoku	Media	Requisitos internos	Completado
RF-07	Interfaz de gráfica Swing con la clase SudokuGUI	Alta	Requisitos internos	Completado
RF-08	Reinicio y cambio de dificultad	Media	Requisitos internos	Completado

### Requisitos no funcionales

ID	Descripción	Categoría	Métrica	Nivel Objetivo	Comentarios
RNF-01	Latencia de validación tras pulsar Enter en GUI	Usabilidad	Tiempo medio entre Enter y respuesta de validación	≤ 100 ms	Mide la fluidez de la interacción en la interfaz

ID	Descripción	Categoría	Métrica	Nivel Objetivo	Comentarios
RNF-02	Tiempo de arranque de la aplicación	Rendimiento	Tiempo hasta que la ventana principal es visible	≤ 2 s	Incluye generación inicial del tablero
RNF-03	Cobertura de pruebas unitarias	Mantenibilidad	% de líneas de código cubiertas por tests	≥ 80 %	Debe incluir lógica de Sudoku y componentes GUI
RNF-04	Consumo máximo de memoria durante ejecución	Rendimiento	Uso máximo de heap al jugar varias rondas	≤ 150 MB	Medido usando herramientas de profiling Java
RNF-05	Compatibilidad multiplataforma	Portabilidad	Plataformas soportadas sin cambios en el código	Windows, macOS, Linux	Requiere Java 11+
RNF-06	Documentación completa con JavaDoc	Mantenibilidad	% de clases y métodos documentados en JavaDoc	≥ 90 %	Incluir ejemplos de uso en README
RNF-07	Robustez ante entradas inválidas	Confiabilidad	% de casos de entrada que no provocan crash	≥ 99 %	Captura y maneja <a href="#">NumberFormatException</a> y otras

## Objetivos

ID	Objetivo SMART	Tipo	Métrica	Fecha Límite	Responsable	Estado
OBJ-01	Generar tableros de Sudoku para "facil", "medio" y "difícil" con ≤30, ≤40 y ≤50 celdas vacías respectivamente	Táctico	% de tableros generados que cumplen el umbral de celdas vacías	2025-05-23	Equipo de Lógica	Terminado

ID	Objetivo SMART	Tipo	Métrica	Fecha Límite	Responsable	Estado
OBJ-02	Completar el algoritmo de backtracking para llenar el tablero en ≤500 ms por tablero	Operativo	Latencia media de <code>fillBoard()</code>	2025-05-23	Equipo de Lógica	Terminado
OBJ-03	Validar movimientos en la GUI en ≤100 ms sin falsos positivos ni negativos	Operativo	Tiempo medio de validación / % de aciertos erróneos	2025-05-23	Equipo de UI/UX	Terminado
OBJ-04	Desplegar interfaz gráfica con filtros de entrada, bordes 3x3, alternancia de color y reinicio "Nuevo"	Táctico	% de requisitos GUI implementados	2025-05-23	Equipo de UI/UX	Terminado
OBJ-05	Proveer interfaz de consola que imprima el tablero tras cada jugada válida y gestione errores de formato	Operativo	% de flujo de consola cubierto / N° de casos de error manejados	2025-05-23	Equipo de Consola	Terminado
OBJ-06	Alcanzar cobertura de pruebas unitarias ≥ 80 % para lógica de Sudoku y generación de tableros	Mantenibilidad	% de líneas de código cubiertas por tests	2025-05-23	Equipo de QA	Terminado
OBJ-07	Documentar 100 % de las clases y métodos con JavaDoc y ejemplos de uso en README	Mantenibilidad	% de elementos con documentación JavaDoc	2025-05-23	Equipo de Documentación	Terminado

ID	Objetivo SMART	Tipo	Métrica	Fecha Límite	Responsable	Estado
OBJ-08	Garantizar compatibilidad en Windows, macOS y Linux con Java 11+ sin ajustes en el código	Estratégico	Nº de plataformas soportadas sin modificaciones	2025-05-23	Equipo de Infraestructura	Terminado

## 2. Diagrama(s) UML (clases y casos de uso)

Diagrama de clases

```
classDiagram

class ISudoku {
    <>
    + int[][] getTablero()
    + void generarTablero(String dificultad)
    + boolean esMovimientoValido(int fila, int col, int val)
    + boolean colocarNumero(int fila, int col, int val)
    + boolean estaResuelto()
    + void mostrarTablero()
}

class IGeneradorSudoku {
    <>
    + static Sudoku generarFacil()
    + static Sudoku generarMedio()
    + static Sudoku generarDificil()
    + static Sudoku generar(String dif)
}

class IJuegoSudoku {
    <>
    + void iniciar()
}

class ISudokuGUI {
    <>
    -void initUI()
}

class Sudoku {
    - int[][] tablero
    - boolean[][] celdasFijas
    - static final int SIZE = 9
    - Random random
    + int[][] getTablero()
    + void generarTablero(String dificultad)
}
```

```

+ boolean esMovimientoValido(int fila, int col, int val)
+ boolean colocarNumero(int fila, int col, int val)
+ boolean estaResuelto()
+ void mostrarTablero()

}

Sudoku ..|> ISudoku

class GeneradorSudoku {
    + static Sudoku generarFacil()
    + static Sudoku generarMedio()
    + static Sudoku generarDificil()
    + static Sudoku generar(String dif)
}

GeneradorSudoku ..|> IGeneradorSudoku
GeneradorSudoku ..> Sudoku : «uses»

class JuegoSudoku {
    + void iniciar()
    + static void main(String[] args)
}

JuegoSudoku ..|> IJuegoSudoku
JuegoSudoku ..> GeneradorSudoku : «uses»
JuegoSudoku ..> Sudoku : «uses»

class SudokuGUI {
    - Sudoku sudoku
    - JTextField[][] celdas
    + SudokuGUI(String dif)
}

SudokuGUI ..|> ISudokuGUI
SudokuGUI o-- Sudoku : «composition»
SudokuGUI ..> GeneradorSudoku : «uses»

```

## Diagrama de casos de uso

```

flowchart LR
%% Actor
J(("👤 Jugador"))

%% Sistema
subgraph "Juego Sudoku"
UC1(("Seleccionar dificultad"))
UC2(("Generar tablero"))
UC3(("Mostrar tablero"))

```

```

UC4(("Introducir movimiento"))
UC5(("Validar movimiento"))
UC6(("Colocar número"))
UC7(("Comprobar solución"))
UC8(("Reiniciar juego / Finalizar"))
end

%% Relaciones Actor -> Casos de Uso
J --> UC1
J --> UC3
J --> UC4
J --> UC8

%% Relaciones entre Casos de Uso
UC1 --> UC2
UC2 --> UC3
UC4 --> UC5
UC5 --> UC6
UC6 --> UC7
UC7 --> UC8

```

### 3. Matriz de trazabilidad

<b>Req. ID</b>	<b>Descripción Req.</b>	<b>Módulo / Clase / Método</b>	<b>Caso de Prueba</b>
RF-01	Generación de tablero 9x9 con casillas vacías según dificultad	GeneradorSudoku / generarFacil(), generarMedio(), generarDificil(), generar(String)	generarFacil(), generarMedio(), generarDificil(), generar()
RF-02	Lógica interna de backtracking para completar tablero válido	Sudoku / fillBoard(), esValido(), shuffleArray()	esMovimientoValido() , colocarNumero(), estaResuelto()
RF-03	Validación de movimientos según reglas de filas, columnas y subcuadrículas	Sudoku / esMovimientoValido(), esValido()	esMovimientoValido()
RF-04	Colocación de números en casillas vacías con verificación de celda fija y rango	Sudoku / colocarNumero()	colocarNumero() (test movimientos inválidos)
RF-05	Verificación de victoria cuando el tablero está completo y válido	Sudoku / estaResuelto()	estaResuelto() (En GUI al resolver)

Req. ID	Descripción Req.	Módulo / Clase / Método	Caso de Prueba
RF-06	Interfaz de consola mostrando tablero en ASCII y leyendo jugadas	JuegoSudoku / iniciar(), main()	mostrarTablero()
RF-07	Interfaz gráfica Swing con grid 9×9, filtros de entrada, bordes y validación Enter	SudokuGUI / initUI(), CellListener.actionPerformed()	(No automatizado: se prueba manualmente en GUI)
RF-08	Reinicio y cambio de dificultad en GUI	SudokuGUI / main(), botón "Nuevo" o menú Archivo→Nuevo	(Manual: se prueba reinicio y cambio de dificultad)
RNF-01	Latencia de validación tras pulsar Enter en GUI	SudokuGUI / CellListener.actionPerformed()	(Medición de rendimiento manual o con herramienta de perfil)
RNF-02	Tiempo de arranque de la aplicación	SudokuGUI / main()	(Medición de tiempo de arranque con profiler)
RNF-03	Cobertura de pruebas unitarias ≥80 %	Todo el código / tests JUnit	Observar comparativa entre pruebas unitarias y métodos y funciones
RNF-04	Consumo máximo de memoria durante ejecución	Sudoku, GeneradorSudoku, SudokuGUI	(Medición con herramienta de profiling de memoria)
RNF-05	Compatibilidad multiplataforma (Windows, macOS, Linux)	Todo el proyecto / Java 11+	(Test de arranque en cada plataforma)
RNF-06	Documentación completa con JavaDoc en ≥90 % de clases y métodos	Todo el código / JavaDoc	Informe de documentación generada
RNF-07	Robustez ante entradas inválidas sin provocar crash	Sudoku / colocarNúmero(), esMovimientoValido(), SudokuGUI/CellListener	Las entradas del usuario están delimitadas dentro del rango correcto

## 4. Pruebas unitarias JUnit

Clase GeneradorSudokuTest

**generarFacil()**

```

@Test
void generarFacil() {
    //Given
    //No hay precondiciones para este test

    //When
    Sudoku sudoku = GeneradorSudoku.generarFacil();
    //Then
    assertNotNull(sudoku, message: "El método generarFacil no debe retornar null");

    //Comprobar que el nivel fácil cumple los requisitos

    int vacias = cuentaVacias(sudoku.getTablero());
    // Fácil permite hasta 30 celdas vacías
    assertTrue( condition: vacias <= 30, message: "El tablero fácil no debe tener más de 30 celdas vacías (encontró " + vacias + ")");
}

```

Comprueba que las celdas vacías a llenar concuerden con las que debe tener la dificultad fácil.

## generarMedio()

```

26
27 @Test
28 void generarMedio() {
29     //Given
30     //No hay precondiciones para este test
31
32     //When
33     Sudoku sudoku = GeneradorSudoku.generarMedio();
34     //Then
35     assertNotNull(sudoku, message: "El método generarFacil no debe retornar null");
36
37     //Comprobar que el nivel medio cumple los requisitos
38     int vacias = cuentaVacias(sudoku.getTablero());
39     // Medio permite hasta 40 celdas vacías
40     assertTrue( condition: vacias <= 40, message: "El tablero fácil no debe tener más de 30 celdas vacías (encontró " + vacias + ")");
41 }

```

Comprueba que las celdas vacías a llenar concuerden con las que debe tener la dificultad media.

## generarDificil()

```

42
43 @Test
44 void generarDificil() {
45     //Given
46     //No hay precondiciones para este test
47
48     //When
49     Sudoku sudoku = GeneradorSudoku.generarDificil();
50     //Then
51     assertNotNull(sudoku, message: "El método generarFacil no debe retornar null");
52
53     //Comprobar que el nivel difícil cumple los requisitos
54
55     int vacias = cuentaVacias(sudoku.getTablero());
56     // Difícil permite hasta 50 celdas vacías
57     assertTrue( condition: vacias <= 50, message: "El tablero fácil no debe tener más de 30 celdas vacías (encontró " + vacias + ")");
58 }

```

Comprueba que las celdas vacías a llenar concuerden con las que debe tener la dificultad difícil.

*Durante los tests se ha usado esta función para evitar la repetición del código*

```
// Método para no repetir este procedimiento en cada test
private int cuentaVacias(int[][][] tablero) { 3 usages
    int cont = 0;
    for (int[] fila : tablero) {
        for (int v : fila) {
            if (v == 0) cont++;
        }
    }
    return cont;
}
```

## generar()

```
59     @Test
60     void generar() {
61         //Dificultad fácil
62         //Given
63         String dificultad = "fAcil";
64         // When
65         Sudoku resultado = GeneradorSudoku.generar(dificultad);
66         // Then
67         assertNotNull(resultado, message: "La conversión a minúsculas debería tratar 'fAcil' como 'facil'");
68
69         //Dificultad media
70         // Given
71         dificultad = "MeDiO";
72         // When
73         resultado = GeneradorSudoku.generar(dificultad);
74         // Then
75         assertNotNull(resultado, message: "La conversión a minúsculas debería tratar 'MeDiO' como 'medio'");
76
77         //Dificultad difícil
78         //Given
79         dificultad = "DiFicil";
80         // When
81         resultado = GeneradorSudoku.generar(dificultad);
82         // Then
83         assertNotNull(resultado, message: "La conversión a minúsculas debería tratar 'DiFicil' como 'difícil'");
84     }
```

Comprueba que se genere el tablero acorde con la dificultad deseada.

## Clase SudokuTest

### getTablero()

```
13     @Test
14     void getTablero() {
15         // Given
16         Sudoku sudoku = new Sudoku();
17
18         // When
19         int[][] tablero = sudoku.getTablero();
20
21         // Then
22         assertNotNull(tablero, message: "El tablero no debe ser null");
23         assertEquals( expected: 9, tablero.length, message: "El tablero debe tener 9 filas");
24         for (int i = 0; i < 9; i++) {
25             assertEquals( expected: 9, tablero[i].length, message: "Cada fila del tablero debe tener 9 columnas");
26             for (int j = 0; j < 9; j++) {
27                 assertEquals( expected: 0, tablero[i][j],
28                             String.format("La posición (%d,%d) debe inicializarse a 0", i, j));
29             }
30         }
31     }
```

Comprueba que el tablero se genere correctamente

### **generarTablero()**

```
33     @Test
34     void generarTablero() {
35         // Given
36         Sudoku sudoku = new Sudoku();
37
38         // When
39
40         // Then
41         assertThrows(NullPointerException.class,
42                     () -> sudoku.generarTablero( dificultad: null),
43                     message: "Pasar null como dificultad debe lanzar NullPointerException");
44     }
```

Comprueba que se genere el tablero

### **esMovimientoValido()**

```

46     @Test
47     void esMovimientoValido() {
48         // Given
49         Sudoku sudoku = new Sudoku(); // tablero vacío, todo 0
50
51         // When
52         boolean resultado = sudoku.esMovimientoValido( fila: 0, col: 0, val: 5);
53
54         // Then
55         assertTrue(resultado, message: "En un tablero vacío, colocar un 5 en (0,0) debe ser válido");
56     }

```

Comprueba que el movimiento es válido

### colocarNumero()

```

58     @Test
59     void colocarNumero() {
60         // Given
61         Sudoku sudoku = new Sudoku(); // tablero vacío, sin celdas fijas
62         ByteArrayOutputStream salida = new ByteArrayOutputStream();
63         PrintStream respaldo = System.out;
64         System.setOut(new PrintStream(salida));
65
66         // When
67         boolean resultado = sudoku.colocarNumero( fila: 0, col: 0, val: 5);
68
69         // Then
70         assertTrue(resultado, message: "Debe devolver true al colocar un número válido en una celda vacía");
71         assertTrue(salida.toString().contains("Número colocado correctamente."),
72                     message: "Debe imprimir 'Número colocado correctamente.'");
73
74         // restaurar salida
75         System.setOut(respaldo);
76     }

```

Comprueba que se coloca el número correctamente cuando el número es válido

### estaResuelto()

```

78     @Test
79     void estaResuelto() {
80         // Given
81         Sudoku sudoku = new Sudoku(); // tablero recién creado, todo a 0
82
83         // When
84         boolean resultado = sudoku.estaResuelto();
85
86         // Then
87         assertFalse(resultado, message: "Un tablero vacío no debe considerarse resuelto");
88     }

```

Comprueba que solo se considere resuelto un tablero relleno (solo se rellena una casilla cuando la respuesta es correcta)

### mostrarTablero()

```
90     @Test
91     void mostrarTablero() {
92         // Given
93         Sudoku sudoku = new Sudoku(); // tablero vacío
94         ByteArrayOutputStream salida = new ByteArrayOutputStream();
95         PrintStream respaldo = System.out;
96         System.setOut(new PrintStream(salida));
97
98         // When
99         sudoku.mostrarTablero();
100
101        // Then
102        String console = salida.toString();
103        // Debe contener la línea horizontal de bloques
104        assertTrue(console.contains("+-----+-----+-----+"),
105                    message: "Debe imprimir separadores horizontales '+-----+-----+'");
106        // Debe contener al menos una línea de celdas con puntos para vacíos
107        assertTrue(console.contains("| . . . |"),
108                    message: "Debe imprimir filas con '| .' seguido de '. .' para celdas vacías");
109
110    }
```

Comprueba de que se imprima en pantalla el sudoku cuando se llame a la función