PROYECTO: SUDOKU

El presente proyecto tiene como objetivo desarrollar una aplicación de escritorio en Java que permita a los usuarios jugar al clásico juego de Sudoku.

Además de la implementación del juego, se deberá realizar un exhaustivo análisis y documentación que abarque desde la definición de objetivos, pasando por la identificación de requisitos (tanto funcionales como no funcionales), hasta la definición de la arquitectura y casos de uso del sistema.

El sistema debe ser capaz de generar tableros con diferentes niveles de dificultad (fácil, medio y difícil), validar las jugadas del usuario en tiempo real y comprobar si el tablero ha sido completado correctamente. Se deberá aplicar conocimientos de programación orientada a objetos, estructuras de datos, control de errores, e interfaces gráficas.

¿Qué es el Sudoku?

El Sudoku es un rompecabezas lógico de colocación de números que se popularizó en Japón en 1986 y se dio a conocer en el ámbito internacional en 2005 a través de los pasatiempos en los periódicos.
El objetivo es rellenar una cuadrícula de 9x9 celdas dividida en subcuadrículas de 3x3, con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas celdas.
No se debe repetir ninguna cifra en una misma fila, columna y subcuadrícula.

Desarrollar una aplicación de escritorio en Java que permita jugar al Sudoku, generando tableros de distintos niveles de dificultad (fácil, medio y difícil), validando las jugadas del usuario en tiempo real y comprobando la solución del tablero.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

El objetivo es completar el tablero con números del 1 al 9, respetando las siguientes
reglas:

1. Regla de Fila; No se pueden repetir números en una misma fila.
2. Regla de Columna; No se pueden repetir números en una misma columna.
3. Regla de Bloque; no se pueden repetir números dentro del bloque 3x3.

A desarrollar:

1. Clase Sudoku

- Atributos:
  - int[][] tablero (matriz principal del tablero).
  - boolean[][] celdasFijas (celdas que no se pueden modificar).
- Métodos clave:
  - generarTablero(String dificultad)

Genera un tablero válido según el nivel ("facil", "medio", "dificil").

  - esMovimientoValido(int fila, int columna, int valor)

Valida los movimientos introducidos por el usuario en tiempo real, asegurando que se cumplen las reglas del Sudoku (sin números repetidos en la misma fila, columna o subcuadrícula). Mostrar mensajes de error claros en caso de que el movimiento no sea válido.

  - colocarNumero(int fila, int columna, int valor)

Inserta un número si el movimiento es válido.

  - estaResuelto()

Verifica si el tablero está completamente resuelto y es válido.

  - mostrarTablero()

Imprime en consola el tablero actual.

2. Clase GeneradorSudoku

- Métodos para generar tableros iniciales válidos con distintas cantidades de
- Fácil: hasta 30 celdas vacías, patrones simples
- Medio: hasta 40 casillas vacías, patrones intermedios
- Difícil: hasta 50 casillas vacías, patrones complejos
- Puede usar técnicas como "backtracking" para resolver y verificar tableros válidos.

3. Clase JuegoSudoku

- Método iniciar () que permite iniciar una partida desde consola:
  - Elegir dificultad.
  - Ver el tablero actual.
- Introducir coordenadas y valores.
- Recibir mensajes de error si el movimiento no es válido.
  - Finalizar si el tablero está completo.

4. Clase SudokuGUI

- Desarrollo de una interfaz gráfica con Swing o JavaFX.
- Permitir la introducción de coordenadas y valores a través de una interfaz (consola y/o gráfica). La interfaz (ya sea de consola o gráfica) debe ser intuitiva y accesible, permitiendo una interacción fácil y sin ambigüedades.

Pruebas Unitarias con JUnit

- Se deben desarrollar pruebas unitarias para todos los métodos clave en cada clase.
- Considerar diferentes tipos de pruebas:
    - Pruebas positivas: Verificar que el sistema se comporta de manera esperada ante entradas correctas.
    - Pruebas negativas: Probar situaciones en las que el usuario introduce datos no válidos (por ejemplo, movimientos en celdas fijas, números fuera del rango permitido, etc.).
    - Pruebas de borde: Validar el comportamiento en condiciones límite, como la validación de los bordes del tablero.

Manejo de Excepciones

- Definir excepciones claras y específicas para gestionar casos de error.
- Algunos ejemplos:
    - SudokuException: Excepción general para errores relacionados con las reglas del juego.
    - MovimientoInvalidoException: Para informar que el movimiento no respeta las reglas del Sudoku.
    - EntradaFueraDeRangoException: Para controlar la entrada de valores que no se encuentran en el rango permitido.

Se recomienda que cada excepción lleve un mensaje descriptivo y, de ser necesario, una solución sugerida o pasos para la corrección del error.

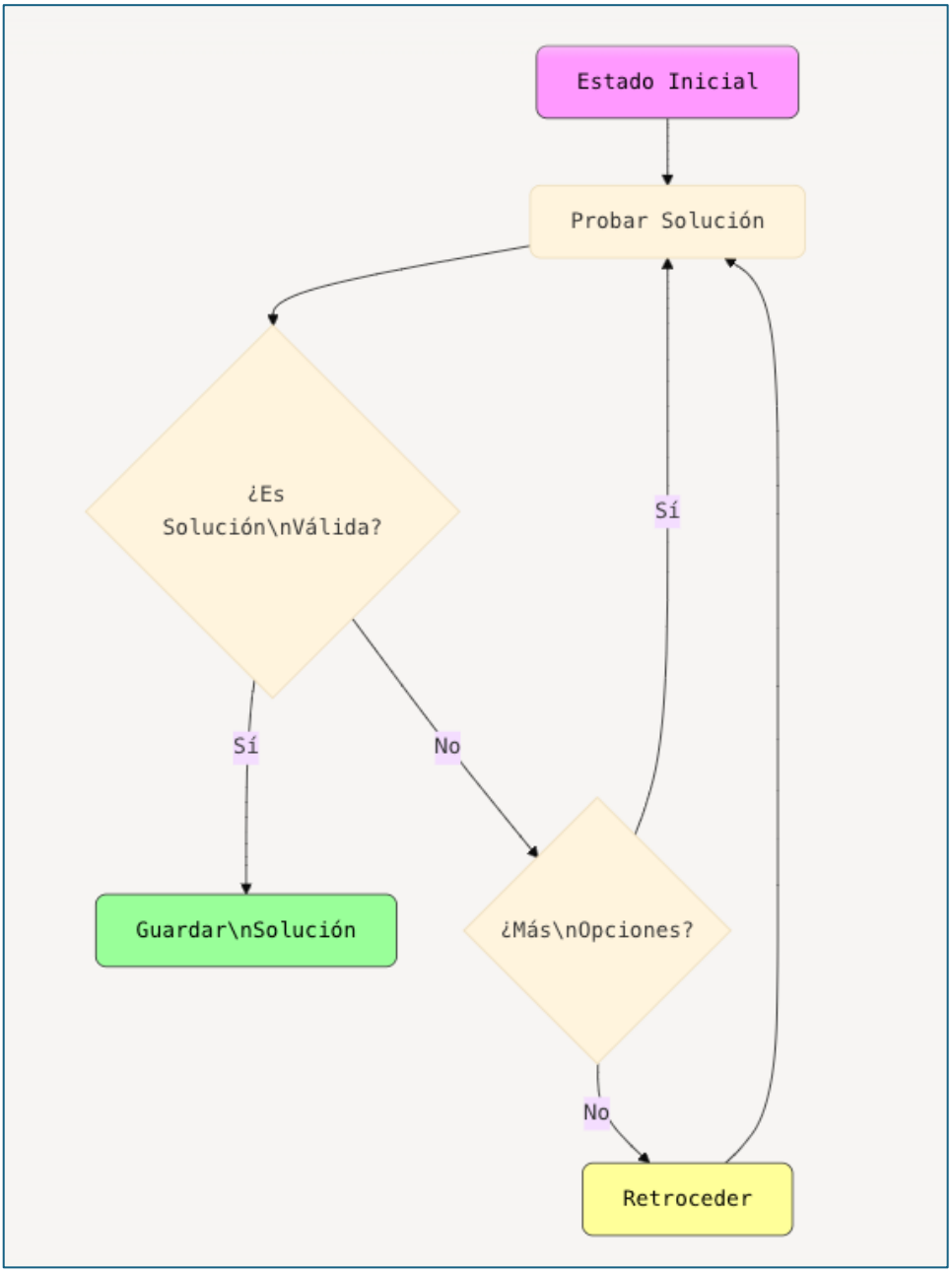
Entrega y Documentación

- **Entrega del Proyecto:**
    - Nombre del archivo: ProyectoSudoku_NombreApellidos.zip y compartir el enlace al repositorio de GitHub.
- **Documentación Adjunta:**
    - Documento de análisis de requerimientos (objetivos, requisitos funcionales y no funcionales).
    - Diagrama(s) UML (clases y casos de uso).
    - Matriz de trazabilidad, que relacione cada requisito (tanto funcional como no funcional) con los módulos, clases y métodos implementados, así como con los casos de prueba correspondientes. Esto permitirá verificar que todos los requisitos se han abordado en el desarrollo y facilita el mantenimiento y futuras modificaciones.
    - Código fuente con comentarios y documentación en Markdown.
    - Conjunto de pruebas unitarias implementadas con JUnit y resultados de la ejecución de estas. Imágenes y explicaciones.

BACKTRAKING

El backtracking es una técnica algorítmica que funciona como un proceso de prueba y error. Imagina que estás resolviendo un rompecabezas: vas probando diferentes piezas, y cuando encuentras que una no funciona, retrocedes (o "backtrack" en inglés) y pruebas otra opción.

Veamos visualmente cómo funciona este proceso:



Veamos en detalle cada paso del diagrama:

1. **Estado Inicial**: Es el punto de partida del problema. Por ejemplo, en un Sudoku, sería el tablero vacío o parcialmente lleno.
2. **Probar Solución**: En cada paso, el algoritmo intenta una posible solución. En el caso del Sudoku, sería colocar un número en una casilla vacía.
3. **¿Es Solución Válida?**: El algoritmo verifica si la solución cumple con todas las reglas:
    - En Sudoku: ¿El número no se repite en la fila, columna ni cuadrado 3x3?
    - En un laberinto: ¿El camino no pasa por paredes?
4. **Guardar Solución**: Cuando encontramos una solución válida, la guardamos. En el Sudoku, esto significaría que hemos encontrado un número que puede permanecer en esa posición.
5. **¿Más Opciones?**: Si la solución actual no es válida, el algoritmo busca otras posibilidades:
    - En Sudoku: probar los números del 1 al 9
    - En un laberinto: probar las diferentes direcciones (arriba, abajo, izquierda, derecha)
6. **Retroceder**: Cuando agotamos todas las opciones sin encontrar una solución válida, el algoritmo "retrocede" al último punto de decisión y prueba otra alternativa. En el Sudoku, esto significa deshacer el último número colocado y probar con otro.

Para entender mejor este proceso, imagina que estás resolviendo un Sudoku:
El algoritmo funcionaría así:

1. Comienza en la primera casilla vacía (marcada con "_")
2. Prueba el número 1
3. Si es válido, continúa con la siguiente casilla
4. Si no es válido, prueba el 2, el 3, etc.
5. Si ninguna opción funciona, retrocede a la casilla anterior y prueba otro número

Este proceso continúa hasta que:

- Se encuentra una solución completa (¡éxito!)
- Se agotan todas las posibilidades (el problema no tiene solución)