**VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY**

**INTERNATIONAL UNIVERSITY**

School of Computer Science & Engineering



**DATA STRUCTURES AND ALGORITHMS PROJECT**

**Topic:** Sudoku Game

**Instructor**: Thái Trung Tín

**Members:**

Nguyễn Trung Anh– ITITWE22122

Mai Tiến Hùng– ITITWE22115

[**I. INTRODUCTION**](#_qn551acbgedm)

[1. Objectives](#_p645zlw8xr8c) 4

[a. Implement Core Game Mechanics](#_yvzq30uoy1ns) 4

[b. Utilize Fundamental Data Structures](#_p8i4fgpmhan3) 4

[c. Implement Advanced Features](#_3ccxveuh39x8) 4

[2. Used tools](#_841by1sx0fa2) 4

[**II. GAMEPLAY AND GAME RULES**](#_7m0w9xqobvqq)

[1. Gameplay](#_fhgkadzhaxv6) 4

[a. Start the Game:](#_1aypr4stfl3k) 4

[b. Enter Numbers:](#_a4wsh0qy7ddd) 5

[c. Mark Mines:](#_7ks6ld8bfvkf) 5

[d. Input Restrictions:](#_igtfs3ibszot) 5

[e. Win Condition:](#_3bg0xpypcwe6) 5

[2. Game rules](#_4qh9q2qwcuwe) 5

[a. Board Structure](#_s0ydew5rglnu) 5

[b. Cell Interaction](#_mxddylr1pgij) 5

[c. Valid Number](#_j7w8jqyzu1wi) 5

[d. Fixed Cells](#_3u503qrr3v09) 5

[e. Validation Logic](#_6j723b877m8a) 5

[f. Endgame](#_9zor1z73dqdy) 5

[3. Levels](#_5c9dbfoxyu3o) 5

a. Easy

b.Medium

c.Hard.

[**III. DETAILS**](#_ebdkxu34u6l5)

[1. Functionality](#_qc7hskjbao6e) 6

[1.1 Main class](#_tmc5bb71mnp7) 6

[1.2 StarScreen class](#_s6co7b3shcum) 6

[1.3 GamePanel class](#_x0rel1ml9vcr) 6

[1.4 SudokuBoard class](#_czzu0qkwpltx) 7

[1.5 SudokuGame class](#_7e887j5w7vl1) 8

[1.6 DigitFilters class](#_k5jns56ih00f) 8

Summary of these functions 9

[**IV. DEMONSTRATION**](#_gwfd9f1ljkgu)

[1. Launching the Game](#_616ox2l83lvj) 10

[2. User Interface](#_771ktcua3cpy) 11

[3. Entering Digits](#_1xrbq9hb4690) 12

[4. Validation Feedback](#_yfloiuqldps3) 13

[5. Winning the Game](#_j1uq7xptyhbh) 16

[6. Difficulty Levels](#_ow8gfsqabhqi) 16

[**V. CONCLUSION**](#_rtiq9tr9qekh)

[1. Summary](#_1rij7bdl36q0) 17

[2. Limitation](#_ykj729oihipx) 17

[3. Evaluation](#_getvptsdudga) 17

[**REFERENCES**](#_7lp5e4qshcbn) 18

# INTRODUCTION

Sudoku is a classic logic-based number puzzle in which players fill a 9×9 grid so that each row, column, and each of the nine 3×3 sub grids contain all digits from 1 to 9 without repetition. This project implements a playable Sudoku game with a graphical interface, input validation, and support for multiple difficulty levels.

## Objectives

### Implement Core Game Mechanics

* Generate valid Sudoku puzzles.
* Allow players to input digits, and validate moves.

### Use OOP Principles:

* Divide the game logic across multiple modular classes such as SudokuBoard, GamePanel, and SudokuGame

### Provide GUI Interactivity:

* Use Java Swing to build an interactive interface with buttons, timers, and styled panels.

## Used tools

* Java JDK 17
* Java Swing (GUI Library)

# GAMEPLAY AND GAME RULES

## Gameplay

### The game begins with a partially-filled 9x9 grid. Players complete the board by entering digits from 1 to 9, ensuring that no duplicate numbers appear in the same row, column, or 3x3 subgrid.

### Start the Game:

The game launches at the Start Screen, where the player can click **“Bắt đầu ván mới”** to start playing with default difficulty (easy). Enter Numbers:

### Players click on cells and input digits (1–9) via keyboard. Only valid digits are allowed.

1. **Win Condition:**

* When the entire grid is filled correctly without any rule violations, the game will win.

1. **Invalid Input Handling:**

* Input is filtered using DigitFilters to allow only numeric values. Game logic also prevents editing of fixed (pre-filled) cells.

## Game rules

### Board Structure

* The Sudoku board is a **9x9 grid**, divided into nine **3x3 subgrids**
* At the start of the game, some cells are **pre-filled** based on the chosen difficulty.

### Cell Interaction

* **Players can click on any editable (empty) cell and type a number from 1 to 9.**

### Pre-filled cells are not editable and are visually distinguished (e.g., with a different background color).

### Fixed Cells

* Fixed cells are part of the initial puzzle and **cannot be modified** during gameplay.
* Editable cells are tracked using a boolean array (isEditable) in the SudokuBoard class.

### Validation Logic

* After each input, the game checks whether the move violates Sudoku rules using isValidMove().
* The player is not prevented from making invalid moves immediately, but validation can be extended for stricter enforcement or visual warnings.
  1. **Endgame**
* The game ends when the entire board is filled correctly and passes all Sudoku rules.

# DETAILS

## Class Description

### Main.java class

#### ****Purpose****:

* Entry point of the application.
* Launches the StartScreen and begins the game flow.

### MineTile class

**Purpose**:

* Displays the starting window with title and a "Start New Game" button.
* Initializes color theme and layout.
* On button click, launches a new GamePanel with a SudokuBoard.

**1.2.1. Constructor**

**Purpose**:

* Initializes frame properties (size, title, layout, center screen).
* Calls initComponents() to prepare GUI elements.
  + 1. **initComponents()**

**Purpose**:

* Creates a panel with styled background and layout.
* Adds the “Start” button and registers its event handler.
  + 1. **startGame(String difficulty)**

**Purpose**:

* Instantiates a new SudokuBoard with given difficulty.
* Creates a new GamePanel to display the board.
* Disposes the StartScreen.

**Time Complexity:**

* O(1) – Constant time to create a SudokuBoard and display the GamePanel.

### GamePanel.java class

### ****Purpose****:

### Main GUI where the Sudoku game board is rendered and played.

* Handles user input.

**1.3.1 Constructor**

**Purpose:**

* Accepts a SudokuBoard and builds a 9x9 grid of JTextFields.
* Sets up the timer and "Back to Menu" button.
  + 1. **setupUI()**

**Purpose**:

* Initializes the Sudoku grid with text fields.
* Disables editing for fixed cells.
* Attaches document filters to ensure valid input.

**Time Complexity:**

* O(n²) – Iterates through a 9x9 grid (n=9) to create text fields and apply filters.
  + 1. **handleBackButton()**

**Purpose**:

* Returns the user to StartScreen and stops the game.

### SudokuBoard.java class

**Purpose:**

1. Generate a valid Sudoku board (solution) using backtracking.
2. Remove some numbers based on difficulty to create a playable puzzle.
3. Validate a number (isValid()) to ensure Sudoku rules are followed.
4. Provide access to both the puzzle (board) and its solution (solution).
5. Allow retrieving values (getValue()) and checking if a cell is fixed (isFixedCell()).

**1.4.1 Constructor**

**Purpose:**

* Initializes a 9x9 board with predefined values based on difficulty.
* Sets up isEditable[][] to track fixed vs input cells.

* + 1. **isValid(int row, int col, int num)**

**Purpose:**

* Checks if placing num at (row, col) violates any rule.
* Returns true if the move is valid.

**Time Complexity**:

* O(n):
* Check row: O(n)
* Check column: O(n)
* Check subgrid (3x3): O(1)
  + Total: O(n), where n = 9.
    1. **solve(int row, int col)**

**Purpose:**

1. **Fills the Sudoku board correctly** following row, column, and 3×3 grid constraints.
2. **Uses Backtracking** to test numbers recursively and undo incorrect placements.
3. **Generates a complete Sudoku solution** for creating a playable puzzle.
4. **Provides a structured way to solve any Sudoku puzzle programmatically**.

**Time Complexity**:

* O() (**x** is the number of unfilled cells, and **k** is the number of possible values that can be placed in each cell. The algorithm will stop once it finds a valid solution)
* **Optimized through pruning** – avoids unnecessary recursive calls by testing **valid moves first**

**How it works:**

* **Base case:** If row == 9, the puzzle is fully solved and returns true.
* **Column reset:** If col == 9, move to the next row (solve(row + 1, 0)).
* **Skip pre-filled cells:** If a number is already placed, move to the next column.
* **Random number selection:** It shuffles [1-9] using randomOrder() to avoid predictable patterns.
* **Validation before placement:** isValid(row, col, num) checks if the number can be placed safely.
* **Recursive solving:** Calls solve(row, col + 1) for the next cell.
* **Backtracking mechanism:** If placement leads to an invalid board, it resets (solution[row][col] = 0) and tries another number.
* **Returns false if no solution is found, triggering a backtrack**.
  + 1. **getBoard() / setBoard()**

**Purpose:**

* Accessor and mutator methods for the board array.

**Time Complexity**:

* O(1): Direct array reference.
  + 1. **getEditableCells()**

**Purpose:**

* Returns the editable cell matrix used by UI.

**Time Complexity:**

* O(1) – Returns a stored array.

**1.5 SudokuGame.java**

**Purpose:**

* Acts as a controller between the GUI and board logic.

**1.5.1 Constructor**

**Purpose:**

* Stores reference to SudokuBoard.
* Optionally coordinates moves or difficulty changes.

**1.5.2 updateCell(int row, int col, int num)**

**Purpose:**

* Updates the board cell with input.
* Validates the move using isValidMove().

**Time Complexity:**

* O(n) – Depends on isValidMove().

**1.6 DigitFilters.java**

**Purpose:**

* Ensures only digits from 1 to 9 are accepted in text fields.

**1.6.1 insertString(...) override**

**Purpose:**

* Overrides document insert behavior.
* Rejects invalid characters or multi-digit inputs.

**Time Complexity:**

* O(1) – Per character input validation.

**Summary of Function Complexities**

|  |  |  |
| --- | --- | --- |
| **Function** | **Purpose** | **Time Complexity** |
| SudokuBoard() | Constructor to initialize the board and editable states | O(n²) |
| isValidMove(row, col, num) | Checks if a number can be legally placed | O(n) |
| isComplete() | Checks if the board is completely and correctly filled | O(n²) |
| setupUI() | Creates and lays out the 9x9 grid in the UI | O(n²) |
| updateTimer() | Updates the timer each second | |  | | --- | |  |  |  | | --- | | O(1) per tick | |
| insertString() (DigitFilters) | Filters user input to ensure only valid digits | O(1) |
| updateCell(row, col, num) | Applies a player move and validates it | O(n) |
| startGame() | Creates a new board and opens the game screen | O(1) |
| main() | Entry point | O(1) |

# DEMONSTRATION

## Launching the Game

* When the program is executed, the user is greeted with a window titled **“Sudoku”**. The interface features a clean layout with a centered button labeled **“Bắt đầu ván mới”**, which means “Start new game”  
  Clicking this button initializes a new game with a pre-defined Sudoku puzzle based on the selected (or default) difficulty.

A screenshot of a computer game

AI-generated content may be incorrect.

*Figure 1. Start Game Screen*

## User Interface

The game interface is neatly divided into two sections:

**Left Section: Sudoku Board**

* A **9×9 grid** represents the game board.
* **Pre-filled numbers** are displayed in black and cannot be edited.
* **Empty cells** are left blank for user input.
* Each cell is clearly outlined, enhancing visibility and usability.

**Top Bar**

* Shows **Difficulty** options: Easy, Medium, Hard (as selectable buttons).
* Displays **Mistake count** (e.g., Mistake: 0/3), indicating how many wrong attempts are allowed before the game ends.

**Right Section: Controls and Input**

* A **3×3 number pad** (buttons labeled 1–9) allows players to input digits into the selected cell.
* Additional buttons include:
* **Check** – Verifies the current board state for mistakes.
* **Hint** – Fills in a correct number (if implemented).
* **Delete** – Clears the selected cell.
* **New Game** – Starts a new puzzle at the selected difficulty level.

A screenshot of a game

AI-generated content may be incorrect.

*Figure 9. Sudoku’s interface*

## Entering Digits

* Players click on an empty cell in the 9×9 grid to select it.
* Then, they click a number (1–9) from the on-screen number pad to place it.

A screenshot of a game

AI-generated content may be incorrect.

*Figure 10. Enter digit’s place*

* If an invalid move is made (e.g., duplicate number in row/column/box), the **mistake counter** is incremented.

A screenshot of a game

AI-generated content may be incorrect.

*Figure 11. The mistake counter increased when we click the wrong digit ( at the above of the enter digit number’s place)*

## Validation Feedback

* After each move, if **Check** is pressed or a mistake is made, the game highlights the mistake.

A screenshot of a game

AI-generated content may be incorrect.

*Figure 12. Valid digit (The blue digit)*

A screenshot of a game

AI-generated content may be incorrect.

*Figure 13. Invalid digit (the red one)*

* The **Mistake Counter** (out of 3) increases. If the player exceeds the limit, the game ends.

A screenshot of a game

AI-generated content may be incorrect.

* + This adds a light punishment mechanic that encourages careful reasoning.

## Winning the Game

* When the entire grid is filled with correct values and no rule violations:
* A **win message** is displayed.
* The board becomes uneditable.
* Players can then start a new game with a different difficulty.

**6. Difficulty Levels**

* The **Easy** level includes many pre-filled cells, making it beginner-friendly.
* **Medium** offers fewer hints and a more balanced challenge.
* **Hard** has minimal clues and is meant for experienced players.
* Switching difficulty resets the board with a new puzzle of that level, creating more excitement for players to try another levels

# CONCLUSION

* + 1. Summary
* The Sudoku project successfully implements the core mechanics of a logic-based number puzzle using Java and Swing. Players can select difficulty levels, input digits using a virtual number pad, and receive feedback on their mistakes. The game's logic validates user inputs and determines win conditions based on traditional Sudoku rules.
* The software structure follows object-oriented programming principles, separating GUI elements from game logic and data structures. Key functionalities such as input filtering, rule validation, and UI control are modularized across classes like SudokuBoard, GamePanel, DigitFilters, and SudokuGame.
  + 1. Limitation
* The current implementation uses **predefined puzzles** rather than generating them dynamically.
* **Hints**, **undo/redo**, and **visual error feedback** could be more robust or fully implemented.
* The game does not yet support **saving/loading progress** or **statistical tracking**.
* There is no **visual highlighting** for incorrect cells beyond the mistake counter.
  + 1. Evaluation
* Applying data structures (2D arrays) and input validation.
* Designing a user-friendly GUI with event-driven programming.
* Structuring code with clean separation of concerns (MVC-style pattern).

# REFERENCES

* **The Brown Box.** (2021, July 29). Build a Sudoku Game in Java [Video]. YouTube. [Sử dụng Thuật Toán để giải trò chơi #Sudoku | #backtracking | Quay Lui](https://www.youtube.com/watch?v=0u0pkBKZoXk&t=148s)