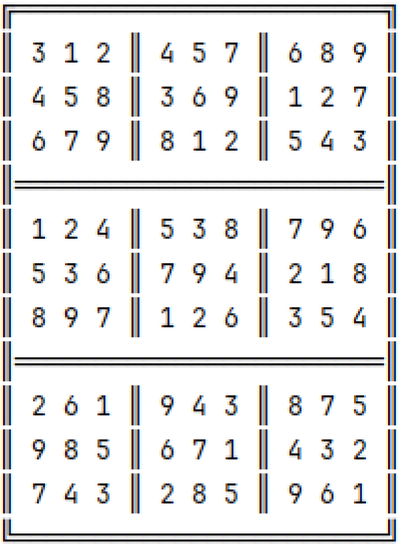
Overview

This project implements a fully functional Sudoku game in Java. It provides users with a Sudoku grid and allows them to solve the puzzle by entering numbers into blank cells. The game includes multiple difficulty levels, an undo functionality, and a validation mechanism to ensure accuracy. It leverages object-oriented programming principles, along with data structures like stacks and queues, to manage the game's state efficiently. The program is modular, making it easy to maintain and extend.



```
╔═══════════════════════╗
║ 3 1 2 ║ 4 5 7 ║ 6 8 9 ║
║ 4 5 8 ║ 3 6 9 ║ 1 2 7 ║
║ 6 7 9 ║ 8 1 2 ║ 5 4 3 ║
║═══════════════════════║
║ 1 2 4 ║ 5 3 8 ║ 7 9 6 ║
║ 5 3 6 ║ 7 9 4 ║ 2 1 8 ║
║ 8 9 7 ║ 1 2 6 ║ 3 5 4 ║
║═══════════════════════║
║ 2 6 1 ║ 9 4 3 ║ 8 7 5 ║
║ 9 8 5 ║ 6 7 1 ║ 4 3 2 ║
║ 7 4 3 ║ 2 8 5 ║ 9 6 1 ║
╚═══════════════════════╝
```

Key Features

- **Dynamic Puzzle Generation**: The Sudoku grid is randomly generated, ensuring uniqueness in every game.

- **Error Handling**: Alerts users when they input invalid entries, such as numbers outside the allowed range or overwriting fixed cells.

- **Undo Feature**: Allows users to revert their last move, providing flexibility in gameplay.
- **Real-Time Validation**: Continuously checks the correctness of user inputs and provides feedback on mistakes.

- **Interactive Console UI**: A user-friendly interface for playing the game through command-line inputs.

- **Efficient Data Structures**: Uses stacks and queues for state management and tracking moves.

- **Difficulty Levels**: Ranging from Beginner to Hard, with varying numbers of pre-filled cells.

Topics of Discrete Mathematics Used

- **Logic and Propositional Statements**:
  The game's validation mechanism is based on logical rules. For example, a valid Sudoku grid satisfies the propositions:
  - A number appears only once in each row `(∀i, j1, j2 (j1 ≠ j2 → grid[i][j1] ≠ grid[i][j2]))`.
  - A number appears only once in each column.
  - A number appears only once in each sub-grid.

  These logical conditions are checked in the **`Validate`** and **`Generate`** classes.

- **Set Theory**:
  Sets are implicitly used to enforce uniqueness of numbers within rows, columns, and sub-grids. The **`forwardCheck()`** method in the **`Generate`** class explicitly manages a 'domain' set to track possible values for each cell.

- **Graph Theory**:
  The Sudoku grid can be represented as a graph where cells are nodes, and edges exist between nodes with constraints (e.g., same row, column, or sub-grid). Ensuring that connected nodes have distinct values relates to concepts like graph coloring.

- **Recursion**:
  The puzzle generation process in the `Generate` class uses recursion to backtrack and find valid configurations, a concept related to recurrence relations in discrete mathematics.

- **Algorithms**:
  The program implements various algorithms:
  - **Backtracking**: Used in puzzle generation to explore and verify valid number placements.
  - **Constraint Propagation**: Iteratively reduces possibilities for each cell by enforcing Sudoku rules, eliminating invalid values to simplify solving.
  - **Validation**: Ensures the correctness of user input dynamically.

- **Combinatorics:**
  Combinatorics is a branch of discrete mathematics that deals with counting, arrangement, and combination of elements within a set under specific rules or constraints. In this program it is used in:

  - **Puzzle Generation:** The placement of numbers in the Sudoku grid requires combinatorial reasoning to ensure that each number appears exactly once in each row, column, and sub-grid.
  - **Cell Removal:** When removing numbers from the grid to create a puzzle of varying difficulty, the program essentially selects a combination of cells to leave blank. The decision-making process, while random, adheres to constraints to ensure solvability.

Coding Aspects and Error Handling

- **Coding Aspects:**
  - **Puzzle Generation**: The `Generate` class uses recursive backtracking and constrain propagation to create a valid Sudoku puzzle, ensuring no conflicts.
  - **Dynamic Cell Removal**: The `RemoveCells` class randomly removes numbers based on the chosen difficulty level.
  - **Validation**: The `Validate` class checks if user inputs match the original puzzle solution and tracks incorrect attempts.
  - **Global State Management**: The `Global` class centralizes shared variables, such as the Sudoku grid, the queue for validations, and the stack for undo operations.

- **Error Handling:**
  - **Invalid Inputs**: Ensures users cannot enter numbers outside the range of 1-9 or overwrite predefined cells.
  - **Boundary Conditions**: Handles edge cases like invalid row/column numbers or exceeding grid boundaries.
  - **Undo Safety**: Prevents users from undoing moves when no moves have been made.
  - **Feedback on Mistakes**: Alerts users to errors in real-time, guiding them to recheck and correct their inputs.

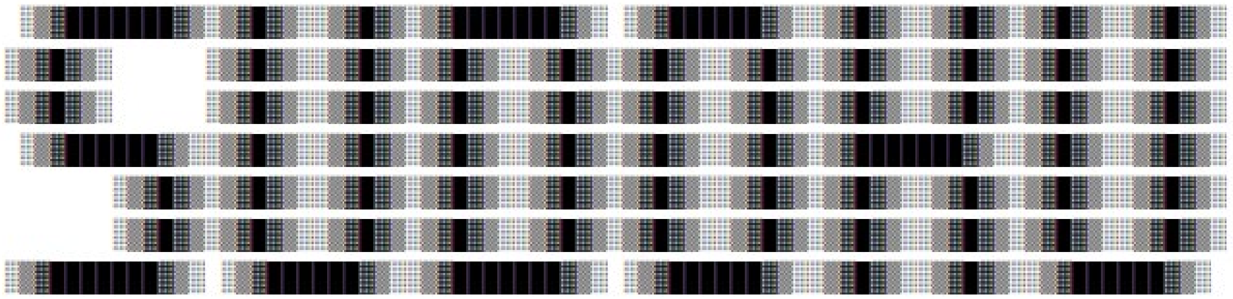
- **Classes Overview:**

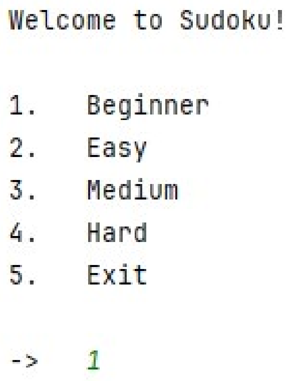
  The program is composed of 11 classes, each serving a specific purpose to ensure modularity and maintainability:

  - `Main`: The entry point that initializes the game through the `ConsoleUI` class.
  - `ConsoleUI`: Handles user interaction, game logic, and inputs via a console-based interface.
  - `Generate`: Responsible for generating a valid Sudoku puzzle using recursive backtracking and constraint propagation.

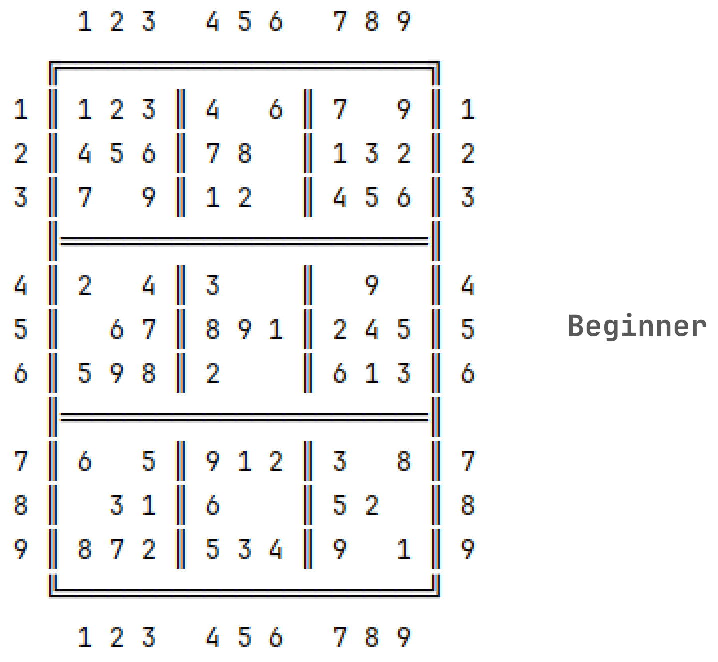
- `RemoveCells`: Manages the removal of numbers from the grid to create puzzles of varying difficulty.
- `Validate`: Checks user inputs against the solution for correctness and tracks mistakes.
- `Input`: Processes user inputs and updates the game state, including storing actions for undo.
- `Undo`: Implements undo functionality by reverting the last user action.
- `Global`: Maintains shared resources like the Sudoku grid, empty grid, stack, and queue.
- `Stack`: Custom implementation of a stack to store user moves for undo operations.
- `Queue`: Custom implementation of a queue to manage validation tasks and errors.
- `Print`: Handles formatted display of the grid and game messages in the console.

Output

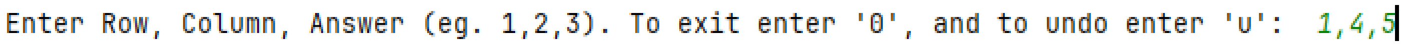




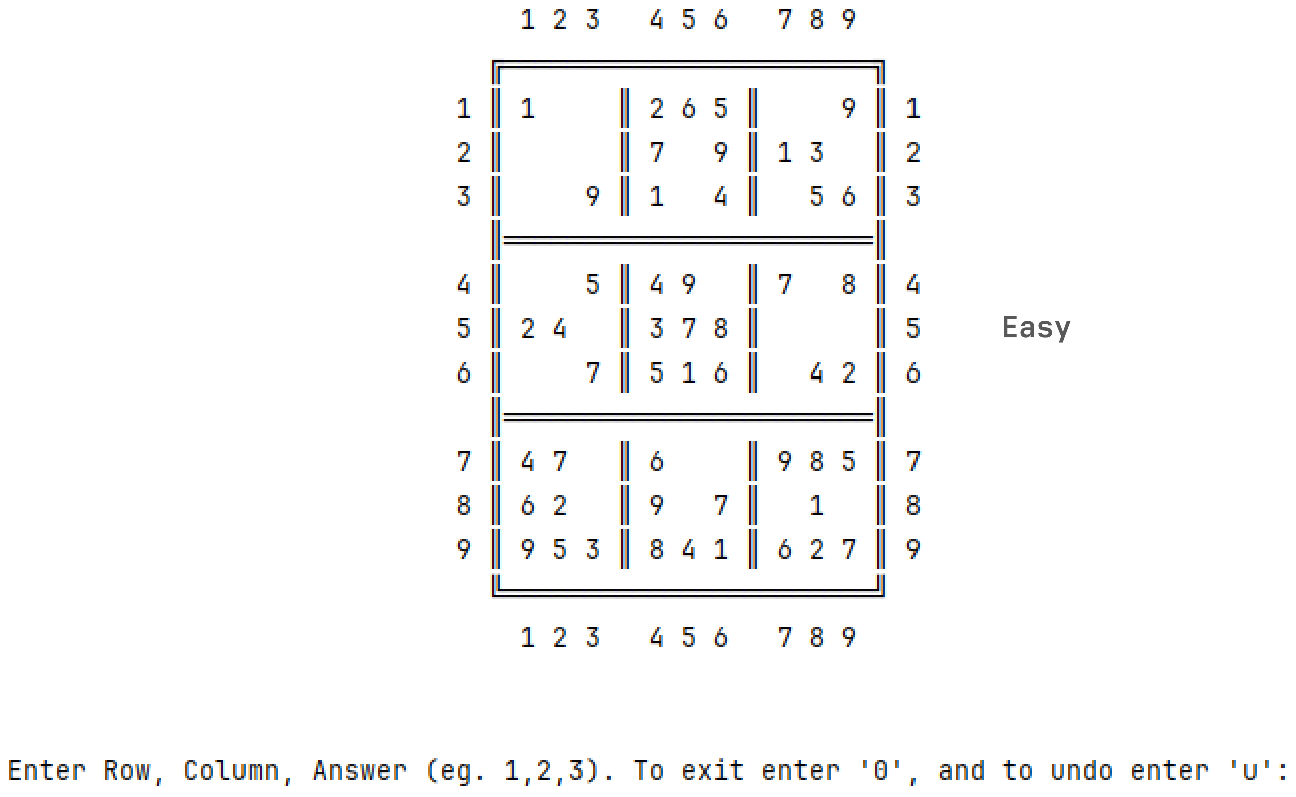
```
Welcome to Sudoku!

1.    Beginner
2.    Easy
3.    Medium
4.    Hard
5.    Exit

->    1
```



```
    1 2 3   4 5 6   7 8 9
  ╔═══════════════════════╗
1 ║ 1 2 3 ║ 4   6 ║ 7   9 ║ 1
2 ║ 4 5 6 ║ 7 8   ║ 1 3 2 ║ 2
3 ║ 7   9 ║ 1 2   ║ 4 5 6 ║ 3
  ║═══════════════════════║
4 ║ 2   4 ║ 3     ║   9   ║ 4
5 ║   6 7 ║ 8 9 1 ║ 2 4 5 ║ 5        Beginner
6 ║ 5 9 8 ║ 2     ║ 6 1 3 ║ 6
  ║═══════════════════════║
7 ║ 6   5 ║ 9 1 2 ║ 3   8 ║ 7
8 ║   3 1 ║ 6     ║ 5 2   ║ 8
9 ║ 8 7 2 ║ 5 3 4 ║ 9   1 ║ 9
  ╚═══════════════════════╝
    1 2 3   4 5 6   7 8 9
```

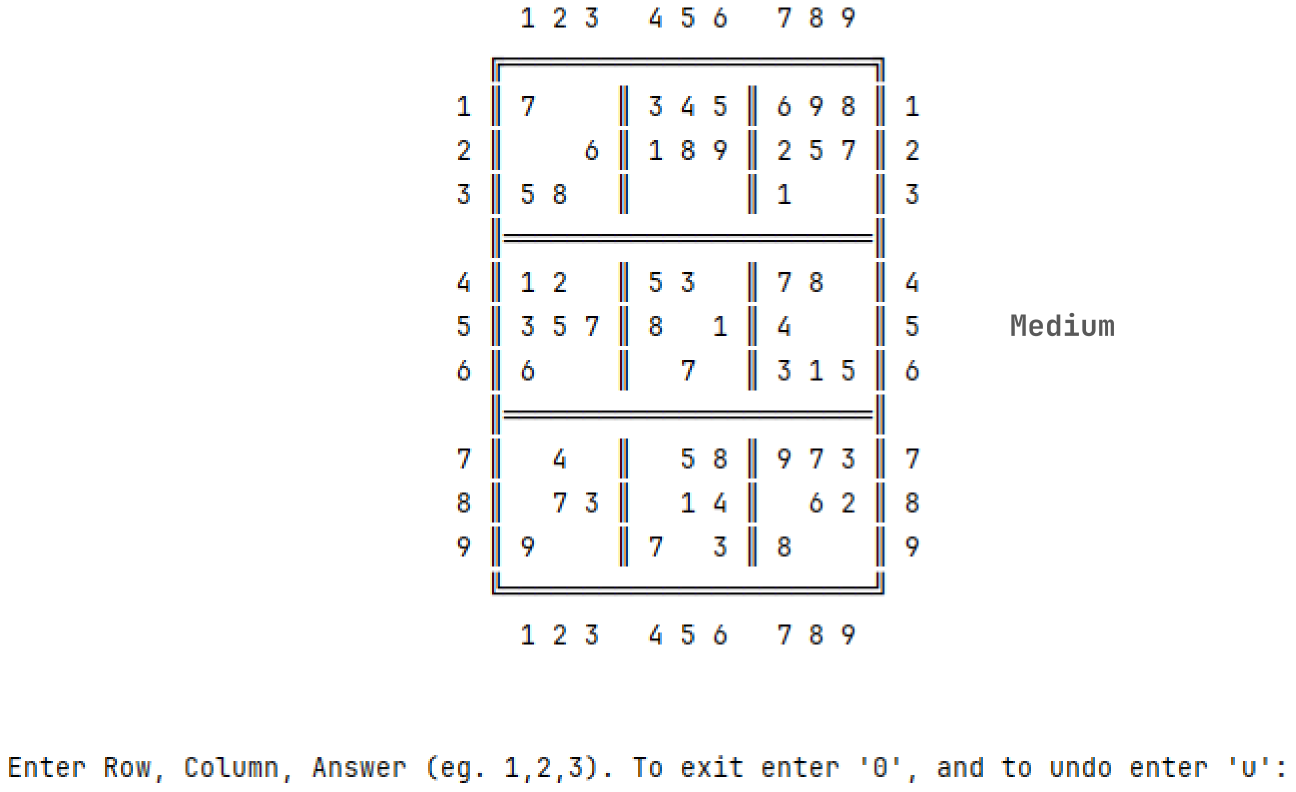


```
Enter Row, Column, Answer (eg. 1,2,3). To exit enter '0', and to undo enter 'u':  1,4,5
```



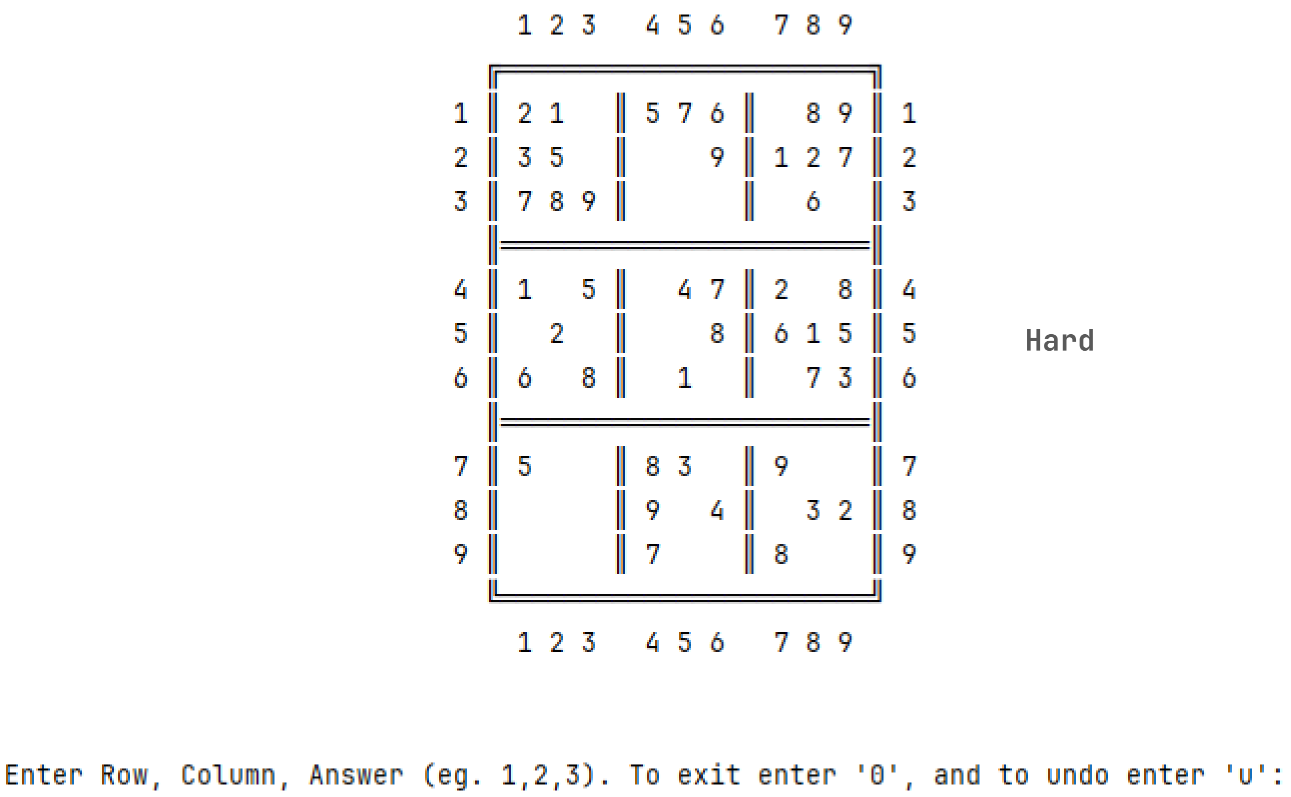
```
    1 2 3   4 5 6   7 8 9
  ╔═══════════════════════╗
1 ║ 1     ║ 2 6 5 ║     9 ║ 1
2 ║       ║ 7   9 ║ 1 3   ║ 2
3 ║     9 ║ 1   4 ║   5 6 ║ 3
  ╠═══════════════════════╣
4 ║     5 ║ 4 9   ║ 7   8 ║ 4
5 ║ 2 4   ║ 3 7 8 ║       ║ 5
6 ║     7 ║ 5 1 6 ║   4 2 ║ 6
  ╠═══════════════════════╣
7 ║ 4 7   ║ 6     ║ 9 8 5 ║ 7
8 ║ 6 2   ║ 9   7 ║   1   ║ 8
9 ║ 9 5 3 ║ 8 4 1 ║ 6 2 7 ║ 9
  ╚═══════════════════════╝
    1 2 3   4 5 6   7 8 9


Enter Row, Column, Answer (eg. 1,2,3). To exit enter '0', and to undo enter 'u':
```

Easy

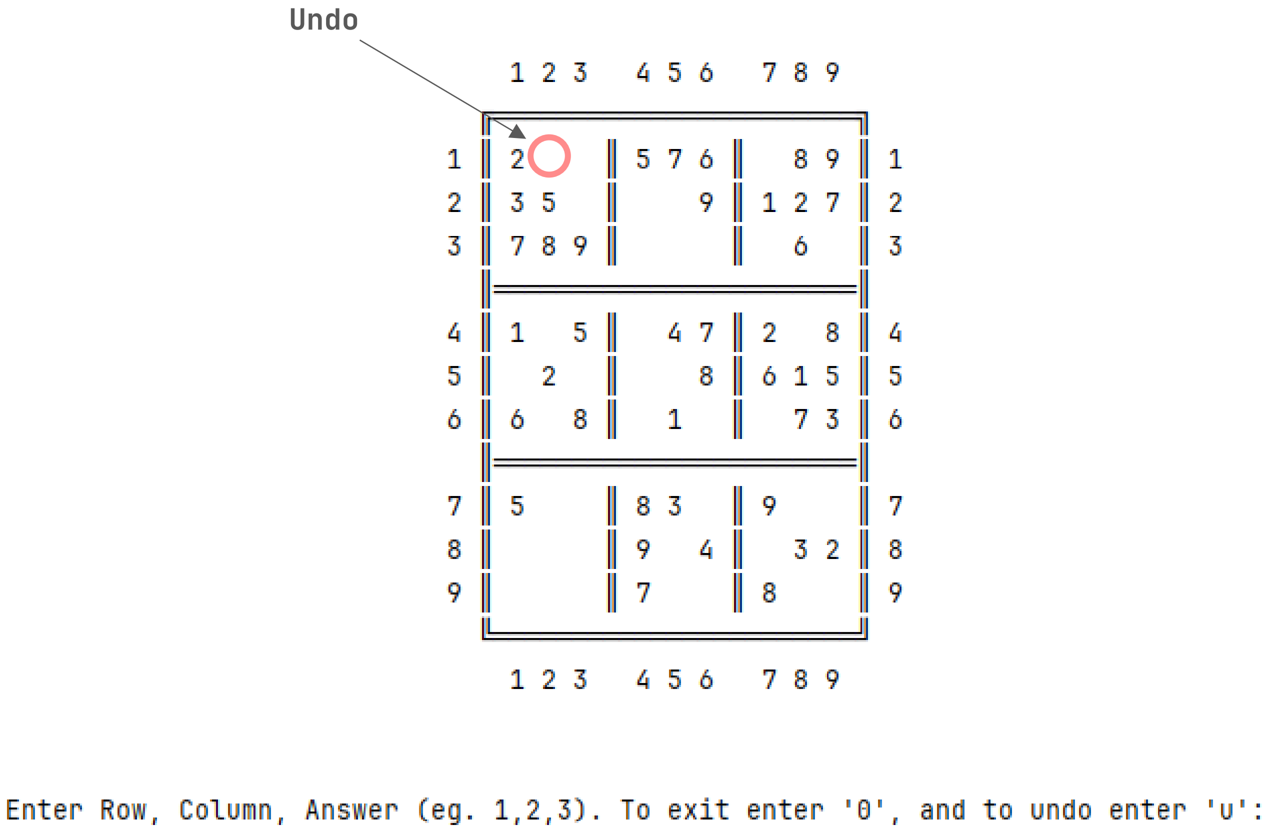


```
    1 2 3   4 5 6   7 8 9
  ╔═══════════════════════╗
1 ║ 7     ║ 3 4 5 ║ 6 9 8 ║ 1
2 ║     6 ║ 1 8 9 ║ 2 5 7 ║ 2
3 ║ 5 8   ║       ║ 1     ║ 3
  ╠═══════════════════════╣
4 ║ 1 2   ║ 5 3   ║ 7 8   ║ 4
5 ║ 3 5 7 ║ 8   1 ║ 4     ║ 5
6 ║ 6     ║   7   ║ 3 1 5 ║ 6
  ╠═══════════════════════╣
7 ║   4   ║   5 8 ║ 9 7 3 ║ 7
8 ║   7 3 ║   1 4 ║   6 2 ║ 8
9 ║ 9     ║ 7   3 ║ 8     ║ 9
  ╚═══════════════════════╝
    1 2 3   4 5 6   7 8 9


Enter Row, Column, Answer (eg. 1,2,3). To exit enter '0', and to undo enter 'u':
```

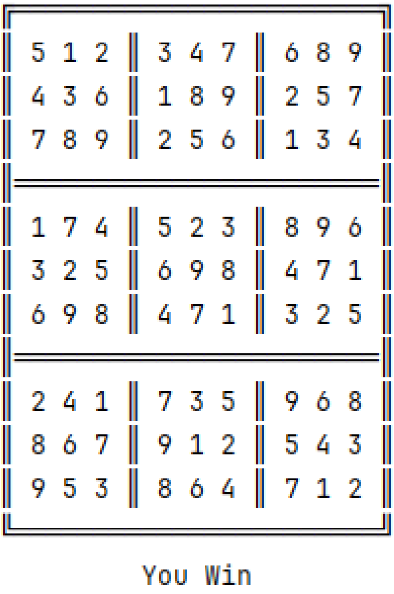
Medium



```
       1 2 3   4 5 6   7 8 9
     ╔═══════════════════════╗
   1 ║ 2 1   ║ 5 7 6 ║   8 9 ║ 1
   2 ║ 3 5   ║     9 ║ 1 2 7 ║ 2
   3 ║ 7 8 9 ║       ║   6   ║ 3
     ║═══════════════════════║
   4 ║ 1   5 ║   4 7 ║ 2   8 ║ 4
   5 ║   2   ║     8 ║ 6 1 5 ║ 5
   6 ║ 6   8 ║   1   ║   7 3 ║ 6
     ║═══════════════════════║
   7 ║ 5     ║ 8 3   ║ 9     ║ 7
   8 ║       ║ 9   4 ║   3 2 ║ 8
   9 ║       ║ 7     ║ 8     ║ 9
     ╚═══════════════════════╝
       1 2 3   4 5 6   7 8 9


Enter Row, Column, Answer (eg. 1,2,3). To exit enter '0', and to undo enter 'u':
```



```
       1 2 3   4 5 6   7 8 9
     ╔═══════════════════════╗
   1 ║ 2     ║ 5 7 6 ║   8 9 ║ 1
   2 ║ 3 5   ║     9 ║ 1 2 7 ║ 2
   3 ║ 7 8 9 ║       ║   6   ║ 3
     ║═══════════════════════║
   4 ║ 1   5 ║   4 7 ║ 2   8 ║ 4
   5 ║   2   ║     8 ║ 6 1 5 ║ 5
   6 ║ 6   8 ║   1   ║   7 3 ║ 6
     ║═══════════════════════║
   7 ║ 5     ║ 8 3   ║ 9     ║ 7
   8 ║       ║ 9   4 ║   3 2 ║ 8
   9 ║       ║ 7     ║ 8     ║ 9
     ╚═══════════════════════╝
       1 2 3   4 5 6   7 8 9


Enter Row, Column, Answer (eg. 1,2,3). To exit enter '0', and to undo enter 'u':
```



```
╔═══════════════════════╗
║ 5 1 2 ║ 3 4 7 ║ 6 8 9 ║
║ 4 3 6 ║ 1 8 9 ║ 2 5 7 ║
║ 7 8 9 ║ 2 5 6 ║ 1 3 4 ║
╠═══════════════════════╣
║ 1 7 4 ║ 5 2 3 ║ 8 9 6 ║
║ 3 2 5 ║ 6 9 8 ║ 4 7 1 ║
║ 6 9 8 ║ 4 7 1 ║ 3 2 5 ║
╠═══════════════════════╣
║ 2 4 1 ║ 7 3 5 ║ 9 6 8 ║
║ 8 6 7 ║ 9 1 2 ║ 5 4 3 ║
║ 9 5 3 ║ 8 6 4 ║ 7 1 2 ║
╚═══════════════════════╝
         You Win
```

Code

```java
import java.util.HashSet;
import java.util.Set;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        new ConsoleUI();
    }
}

public class ConsoleUI {

    ConsoleUI() {

        Scanner s = new Scanner(System.in);
        Global g = new Global();
        Print p = new Print();
        Input in = new Input();
        Undo u = new Undo();
        Validate v = new Validate();
        RemoveCells r = new RemoveCells();
        boolean found  = false;

        p.title3();
        System.out.print(
                "\t\t\t\t\t\t\t\tWelcome to Sudoku!" +
                        "\n\n\t\t\t\t\t\t\t\t1.   Beginner" +
                        "\n\t\t\t\t\t\t\t\t2.   Easy" +
                        "\n\t\t\t\t\t\t\t\t3.   Medium" +
                        "\n\t\t\t\t\t\t\t\t4.   Hard" +
                        "\n\t\t\t\t\t\t\t\t5.   Exit" +
                        "\n\n\t\t\t\t\t\t\t\t->   "
        );
        int choice = s.nextInt();

        switch (choice) {
            case 0 -> r.remove(0);
            case 1 -> r.remove(1);
            case 2 -> r.remove(2);
            case 3 -> r.remove(3);
            case 4 -> r.remove(4);
            case 5 -> {return;}
            case 6 -> r.remove(6);
        }
        System.out.println("\n\n\n");

        while (true) {

            System.out.println();
            p.format1();

            System.out.print("\n\nEnter Row, Column, Answer (eg. 1,2,3). To exit enter '0', and 
to undo enter 'u':  ");
            String answer = s.next();
            String[] ijv = answer.split(",");
            if (ijv[0].equals("u")) {
                u.redo();

            } else if (ijv[0].equals("p")) {
                for (int i = 0; i < 9; i++) {
                    for (int j = 0; j < 9; j++) {
                        System.out.print(g.grid[i][j] + " ");
                    }
                    System.out.println();
                }
                System.out.println();

                for (int i = 0; i < 9; i++) {
                    for (int j = 0; j < 9; j++) {
                        System.out.print(g.emptyGrid[i][j] + " ");
                    }
                    System.out.println();
                }
                continue;

            } else if (ijv[0].equals("q")) {
                g.queue.print();
                continue;

            } else if (Integer.parseInt(ijv[0]) == 0) {
                return;

            } else if (g.emptyGrid[Integer.parseInt(ijv[0]) - 1][Integer.parseInt(ijv[1]) - 1]
== 0) {
                System.out.println("\nCannot change an anchor.");
                continue;

            } else if (Integer.parseInt(ijv[0]) > 9 || Integer.parseInt(ijv[0]) < 1 ||
Integer.parseInt(ijv[1]) > 9 || Integer.parseInt(ijv[1]) < 1){
                System.out.println("\nThere are only 9 rows and 9 columns.");
                continue;

            } else if (Integer.parseInt(ijv[2]) > 9 || Integer.parseInt(ijv[2]) < 1){
                System.out.println("\nEnter number between 1-9.");
                continue;

            } else {
                in.enter(Integer.parseInt(ijv[0]) - 1, Integer.parseInt(ijv[1]) - 1,
Integer.parseInt(ijv[2]));
            }

            for (int i = 0; i < 9; i++) {
                int j;

                for (j = 0; j < 9; j++) {
                    if (g.emptyGrid[i][j] == -1) {
                        found = true;
                        break;
                    }
                    found = false;
                }

                if (j != 9) {
                    break;
                }
            }

            if (found) {
```

```java
                    continue;
                }

                if (!found) {
                    if (v.check() == 0) {
                        System.out.println("\n\n");
                        p.format2();
                        System.out.print("\t\t\t\t\t\t\t\t\t   You Win");
                        return;
                    }

                    g.recheck = true;
                    System.out.println("\nThere are " + g.incorrect + " mistake, recheck.");
                }
            }
        }
    }


    public class Print {

        private static int count = 1;
        private Global g;

        Print() {
            g = new Global();
        }

        public void title3() {
            System.out.println(
          "\n" +
          "\t ░▒▓███████▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓███████▓▒░ ░▒▓██████▓▒░░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░  \n" +
          "\t░▒▓█▓▒░      ░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░  \n" +
          "\t░▒▓█▓▒░      ░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░  \n" +
          "\t ░▒▓██████▓▒░░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓███████▓▒░░▒▓█▓▒░░▒▓█▓▒░  \n" +
          "\t       ░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░  \n" +
          "\t       ░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░▒▓█▓▒░░▒▓█▓▒░  \n" +
          "\t░▒▓███████▓▒░ ░▒▓██████▓▒░░▒▓███████▓▒░ ░▒▓██████▓▒░░▒▓█▓▒░░▒▓█▓▒░░▒▓██████▓▒░   \n" +
          "\t                                                                                 \n" +
          "\t                                                                                 \n"
            );
        }


        public void format1() {

            count = 1;

            System.out.println("\t\t\t\t\t\t\t    1 2 3   4 5 6   7 8 9");
            System.out.println("\t\t\t\t\t\t\t  ╔══════════════════════╗");

            for (int i = 0; i < 9; i++) {

                System.out.print("\t\t\t\t\t\t\t" + count + " ║ ");

                for (int j = 0; j < 9; j++) {

                    if (g.emptyGrid[i][j] == 0) {
                        System.out.print(g.grid[i][j] + " ");
                    } else if (g.emptyGrid[i][j] == -1){
                        System.out.print("  ");
                    } else {
                        System.out.print(g.emptyGrid[i][j] + " ");
                    }
                    if ((j + 1) % 3 == 0 && j < 8) {
                        System.out.print("║ ");
                    }
                }

                System.out.println("║ " + count);
                count++;

                if ((i + 1) % 3 == 0 && i < 8) {
                    System.out.println("\t\t\t\t\t\t\t  ║════════════════════════║");
                }
            }

            System.out.println("\t\t\t\t\t\t\t  ╚════════════════════════╝");
            System.out.println("\t\t\t\t\t\t\t    1 2 3   4 5 6   7 8 9");
        }


        public void format2() {

            count = 1;

            System.out.println("\t\t\t\t\t\t\t  ╔════════════════════════╗");

            for (int i = 0; i < 9; i++) {

                System.out.print("\t\t\t\t\t\t\t  ║ ");

                for (int j = 0; j < 9; j++) {

                    System.out.print(g.grid[i][j] + " ");

                    if ((j + 1) % 3 == 0 && j < 8) {
                        System.out.print("║ ");
                    }
                }

                System.out.println("║");
                count++;

                if ((i + 1) % 3 == 0 && i < 8) {
                    System.out.println("\t\t\t\t\t\t\t  ║════════════════════════║");
                }
            }

            System.out.println("\t\t\t\t\t\t\t  ╚════════════════════════╝");
        }
    }

    public class Global {

        public static int[][] grid;
        public static int[][] emptyGrid;
        public static Stack stack;
        public static Queue queue;
        public static int incorrect;
        public static boolean recheck;

        Global() {
            grid = new int[9][9];
            emptyGrid = new int[9][9];
            stack = new Stack(0);
            queue = new Queue(0);
            incorrect = 0;
            recheck = false;
            grid = new Generate().createPuzzle();
        }
    }

    public class Generate {

        private int[][] grid;

        Generate() {
            grid = new int[9][9];
        }

        public int[][] createPuzzle() {

            int i = 0;

            grid[0][0] = (int) (Math.random() * 9);
            grid[8][8] = (int) (Math.random() * 9);

            while (i < 5) {

                int row = (int) (Math.random() * 9);
                int col = (int) (Math.random() * 9);
                int num = (int) (Math.random() * 9);

                if (check(row, col, num)) {
                    grid[row][col] = num;
                    i++;
                }
            }

            generatePuzzle();
            return grid;
        }

        private boolean generatePuzzle() {

            int[] emptyCell = emptyCell();

            if (emptyCell == null) {
                return true;
            }

            int i = emptyCell[0];
            int j = emptyCell[1];

            for (int num = 1; num < 10; num++) {

                if (check(i, j, num)) {
                    grid[i][j] = num;

                    if (forwardCheck() && generatePuzzle()) {
                        return true;
                    }
                    grid[i][j] = 0;
                }
            }
            return false;
        }

        private boolean forwardCheck() {

            for (int i = 0; i < 9; i++) {
                for (int j = 0; j < 9; j++) {

                    if (grid[i][j] == 0) {

                        Set<Integer> domain = new HashSet<>();

                        for (int k = 1; k < 10; k++) {
                            domain.add(k);
                        }

                        for (int k = 0; k < 9; k++) {
                            domain.remove(grid[i][k]);
                            domain.remove(grid[k][j]);
                        }

                        int startRow = (i / 3) * 3;
                        int startCol = (j / 3) * 3;

                        for (int l = 0; l < 3; l++) {
                            for (int m = 0; m < 3; m++) {
                                domain.remove(grid[startRow + l][startCol + m]);
                            }
                        }

                        if (domain.isEmpty()) {
                            return false;
                        }
                    }
                }
            }
            return true;
        }

        private boolean check(int row, int col, int num) {

            for (int i = 0; i < 9; i++) {
                if (grid[row][i] == num || grid[i][col] == num) {
                    return false;
                }
            }

            int startRow = (row / 3) * 3;
            int startCol = (col / 3) * 3;

            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {

                    if (grid[startRow + i][startCol + j] == num) {
                        return false;
                    }
                }
            }
            return true;
        }

        private int[] emptyCell() {

            for (int row = 0; row < 9; row++) {
                for (int col = 0; col < 9; col++) {

                    if (grid[row][col] == 0) {
                        return new int[]{row, col};
                    }
                }
            }
            return null;
        }
    }

    public class Input {

        private Global g;

        Input() {
            g = new Global();
        }

        public void enter(int i, int j, int value) {

            g.stack.push(g.emptyGrid[i][j]);
            g.emptyGrid[i][j] = value;
            g.stack.push(j);
            g.stack.push(i);

            if (g.recheck) {
                g.queue.enqueue(i);
                g.queue.enqueue(j);
                g.incorrect++;
            }
        }
    }

    public class Queue {

        int index;
        private Queue head;
        private Queue tail;
        private Queue next;
        public int size;

        Queue(int index) {
            this.index = index;
        }

        public void enqueue(int index) {

            Queue temp = new Queue(index);
            if (head == null) {
                head = temp;
                tail = temp;
                head.next = tail;
                size++;
                return;
            }

            tail.next = temp;
            tail = temp;
            size++;
        }

        public int dequeue() {
            int popped = head.index;
            head = head.next;
            size--;
            return popped;
        }

        public void print() {

            Queue temp = head;

            while (temp != null) {
                System.out.print(temp.index + ", ");
                temp = temp.next;
                System.out.println(temp.index);
                temp = temp.next;
            }
        }
    }

    public class Stack {

        private int index;
        private Stack head;
        private Stack next;

        Stack(int index) {
            this.index = index;
        }

        public void push(int index) {

            Stack temp = new Stack(index);

            if (head == null) {
                head = temp;
                return;
            }

            temp.next = head;
            head = temp;
        }

        public int pop() {
            int popped = peek();
```

```java
        head = head.next;
        return popped;
    }

    public int peek() {
        if (head == null) {
            return -2;
        }
        return head.index;
    }
}

public class Undo {

    private Global g;

    Undo() {
        g = new Global();
    }

    public void redo() {

        if (g.stack.peek() == -2) {
            return;
        }

        int i = g.stack.pop();
        int j = g.stack.pop();
        int value = g.stack.pop();

        g.emptyGrid[i][j] = value;

        if (g.recheck) {
            g.queue.enqueue(i);
            g.queue.enqueue(j);
            g.incorrect++;
        }
    }
}

public class RemoveCells {

    private Global g;

    RemoveCells() {
        g = new Global();
    }

    public void remove(int level) {

        switch (level) {
            case 0 -> g.incorrect = 1;
            case 1 -> g.incorrect = 24;
            case 2 -> g.incorrect = 37;
            case 3 -> g.incorrect = 46;
            case 4 -> g.incorrect = 55;
            case 6 -> g.incorrect = 2;
        }
        for (int i = 0; i < g.incorrect; i++) {
            int row, col;

            do {
                row = (int) (Math.random() * 9);
                col = (int) (Math.random() * 9);

            } while (g.emptyGrid[row][col] == 1);

            g.emptyGrid[row][col] = -1;
            g.queue.enqueue(row);
            g.queue.enqueue(col);
        }
    }
}

public class Validate {

    private Global g;

    Validate() {
        g = new Global();
    }

    public int check() {

        int l = g.queue.size;

        for (int k = 0; k < l; k = k + 2) {

            int i = g.queue.dequeue();
            int j = g.queue.dequeue();

            if (g.grid[i][j] != g.emptyGrid[i][j]) {
                g.queue.enqueue(i);
                g.queue.enqueue(j);

            } else {
                g.incorrect--;
            }
        }
        return g.incorrect;
    }
}
```