



2ECDE55-SCRIPTING LANGUAGE
SPECIAL ASSIGNMENT

TOPIC: SUDOKU PUZZLE SOLVER

Submitted by:

NAME	ROLL NO.
PRATHAM CHHAJED	21BEC020
CHIRAG RAJWANI	21BEC021

Submitted to:

Prof. Vaishali H. Dhare

Department of Electronics & Communications Engineering, IT-NU

Abstract :-

The objective of this project is to implement a Sudoku solver in the Perl programming language using a backtracking algorithm. The solver takes an incomplete Sudoku puzzle as input and attempts to fill in the empty cells, ensuring that the completed puzzle adheres to the rules of Sudoku.

User's only task is to provide command terminal with a command "perl sudokupuzzle.pl" and then giving a random 9x9 sudoku quiz as an input to get the solved output in result.

Introduction :-

Sudoku is a popular number puzzle game that requires logical thinking and problem-solving skills. The puzzle begins with some cells already filled, and the solver's(our programs') task is to complete the grid as per the rules of the game. Also, report mentions of a real life example of the gameplay in form of an already popularly played sudoku puzzle game present online compared with our encoded puzzle solver for a same given quiz output; verifying the accuracy of our code.

The game has been coded in 'perl 5, version 36, subversion 0 (v5.36.0) built for x86_64-linux-gnu-thread-multi' using the nano text editor and the outputs of the program are observed over the command terminal.

➤ Following points form the basic motivation for constructing this kind of a code→

1.To provide the beginners or rookies as an hint to much higher difficulty level sudoku puzzles which they could create as per their own wish and then can use the solved output wherever they feel stuck or in a challenging state which enhances the overall user experience and encourages continued engagement.

2.Using such an ease going form the game would aid in maintaining the buzz and interest level of first time explorers of online games,

wherein they could be attracted to such complex brainstorming and developing games rather than time consuming no brainers.

Implementation :-

❖ Working and Initialization →

The Sudoku solver is implemented in Perl, a versatile scripting language. The code is organized into several subroutines to improve readability and modularity, some of which are as follows:

➤ Initialize the Sudoku board

```
my $sudoku = [  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0],  
];
```

- **print_sudoku**: This subroutine prints the current state of the Sudoku grid to the console.
- **solve_sudoku**: The core of the solver, this recursive subroutine attempts to solve the puzzle by iteratively placing numbers in empty cells and backtracking when necessary.
- **is_valid_move**: Checks if a number can be legally placed in a given cell by verifying its absence in the corresponding row, column, and 3x3 subgrid.

- in_row, in_col, in_box: Helper subroutines that check if a given number is present in a specified row, column, or subgrid.

❖ Usage →

The user is prompted to enter the initial Sudoku puzzle row by row, using '0' to represent empty cells. The solver then attempts to find a solution using the implemented backtracking algorithm. If a solution is found, the solved Sudoku grid is displayed; otherwise, a message indicating that no solution exists is printed.

The initial output which could be expected by a user is represented below:-

Enter the Sudoku puzzle row by row. Use '0' for empty cells.

Enter row 0: 5 3 0 0 7 0 0 0 0

Enter row 1: 6 0 0 1 9 5 0 0 0

Enter row 2: 0 9 8 0 0 0 0 6 0

Enter row 3: 8 0 0 0 6 0 0 0 3

Enter row 4: 4 0 0 8 0 3 0 0 1

Enter row 5: 7 0 0 0 2 0 0 0 6

Enter row 6: 0 6 0 0 0 0 2 8 0

Enter row 7: 0 0 0 4 1 9 0 0 5

Enter row 8: 0 0 0 0 8 0 0 7 9

Solved Sudoku:

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

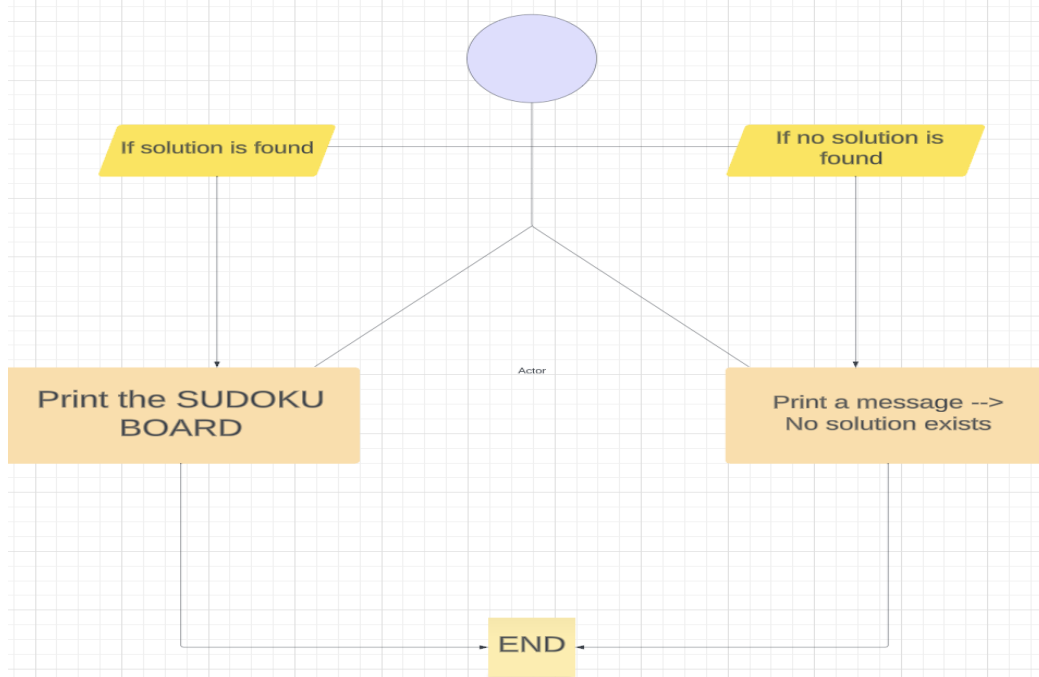
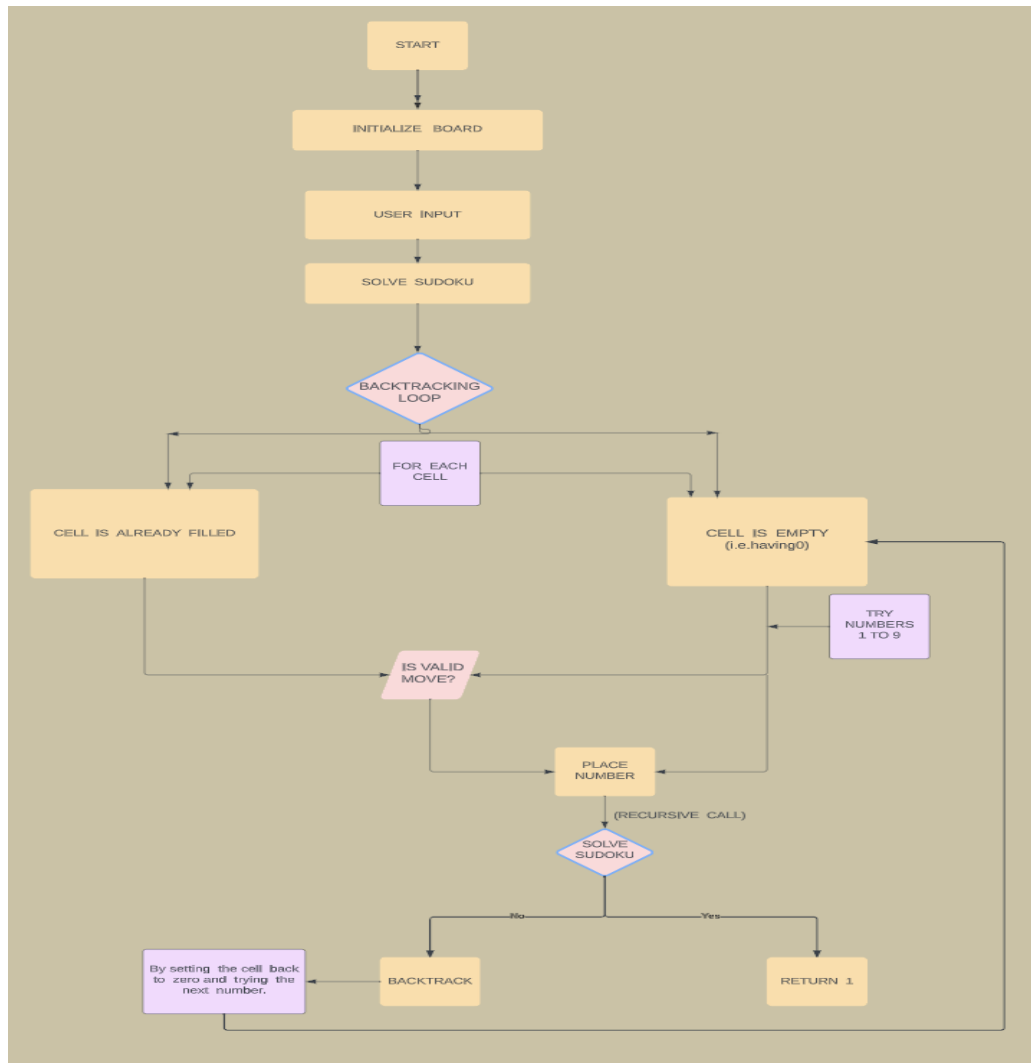
7 1 3 9 2 4 8 5 6

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

Flowchart :-



PERL CODE :-

```
#!/usr/bin/perl
use strict;
use warnings;

# Initialize the Sudoku board
my $sudoku = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
];

# Prompt the user to enter the Sudoku puzzle
print "Enter the Sudoku puzzle row by row. Use '0' for empty cells.\n";
for my $row (0..8) {
    print "Enter row $row: ";
    my $input = <STDIN>;
    chomp $input;
    my @values = split /\s+/, $input;
    for my $col (0..8) {
        $sudoku->[$row][$col] = $values[$col];
    }
}
```

```
# Solve the Sudoku puzzle
if (solve_sudoku($sudoku)) {
    print "\nSolved Sudoku:\n";
    print_sudoku($sudoku);
} else {
    print "\nNo solution found.\n";
}

sub print_sudoku {
    my $board = shift;
    for my $row (@$board) {
        for my $cell (@$row) {
            print "$cell ";
        }
        print "\n";
    }
}

sub solve_sudoku {
    my ($board) = @_;
    for my $row (0..8) {
        for my $col (0..8) {
            unless ($board->[$row][$col]) {
                for my $num (1..9) {
                    if (is_valid_move($board, $row, $col, $num)) {
                        $board->[$row][$col] = $num;
                        if (solve_sudoku($board)) {
                            return 1;
                        }
                    }
                }
            }
        }
    }
    return 0;
}
```

```

        } else {
            $board->[$row][$col] = 0;
        }
    }
}
return 0;
}
}
return 1;
}

sub is_valid_move {
    my ($board, $row, $col, $num) = @_;
    return (
        !in_row($board, $row, $num) &&
        !in_col($board, $col, $num) &&
        !in_box($board, $row - $row % 3, $col - $col % 3, $num)
    );
}

sub in_row {
    my ($board, $row, $num) = @_;
    return grep { $_ == $num } @{$board->[$row]};
}

sub in_col {

```

```

    sub in_col {
        my ($board, $col, $num) = @_;
        return grep { $_ == $num } map { $_->[$col] } @$board;
    }

    sub in_box {
        my ($board, $start_row, $start_col, $num) = @_;
        for my $row (0..2) {
            for my $col (0..2) {
                return 1 if $board->[$row + $start_row][$col + $start_col] == $num;
            }
        }
        return 0;
    }
}

```

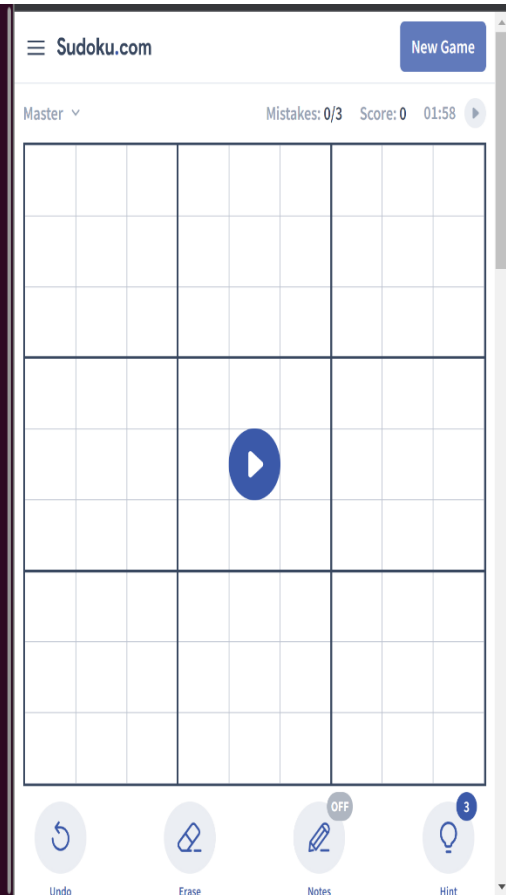
OUTPUTS:

```
chirag-21@chirag-21-VirtualBox:~$ vi sudokupuzzle.pl
chirag-21@chirag-21-VirtualBox:~$ perl sudokupuzzle.pl
Enter the Sudoku puzzle row by row. Use '0' for empty cells.
Enter row 0: 9 0 0 0 0 0 0 0 4
Enter row 1: 0 3 0 0 7 0 6 9 0
Enter row 2: 0 0 2 8 0 0 0 0 0
Enter row 3: 0 5 0 0 0 0 0 0 1
Enter row 4: 0 0 0 0 4 0 0 0 3
Enter row 5: 8 0 0 7 0 0 4 5 0
Enter row 6: 3 0 0 0 0 9 0 0 0
Enter row 7: 0 0 0 0 0 0 0 1 0
Enter row 8: 0 9 0 0 6 0 5 7 0
```

Solved Sudoku:

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

```
chirag-21@chirag-21-VirtualBox:~$
```

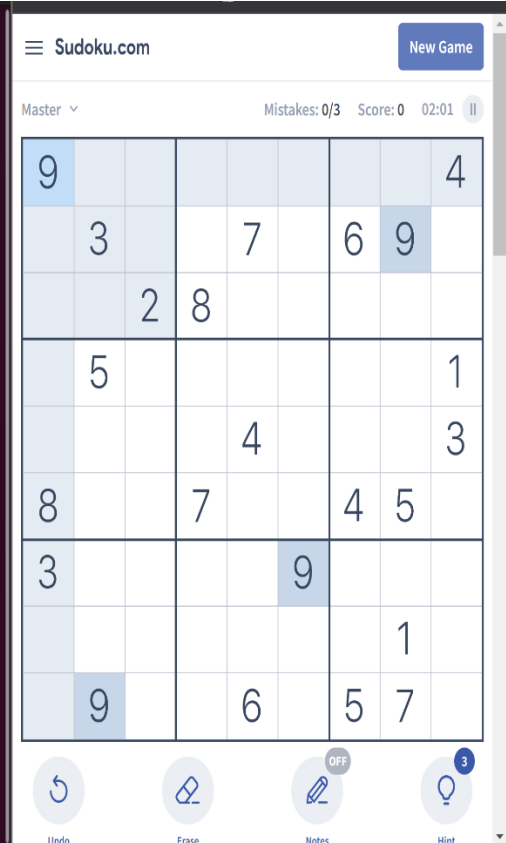


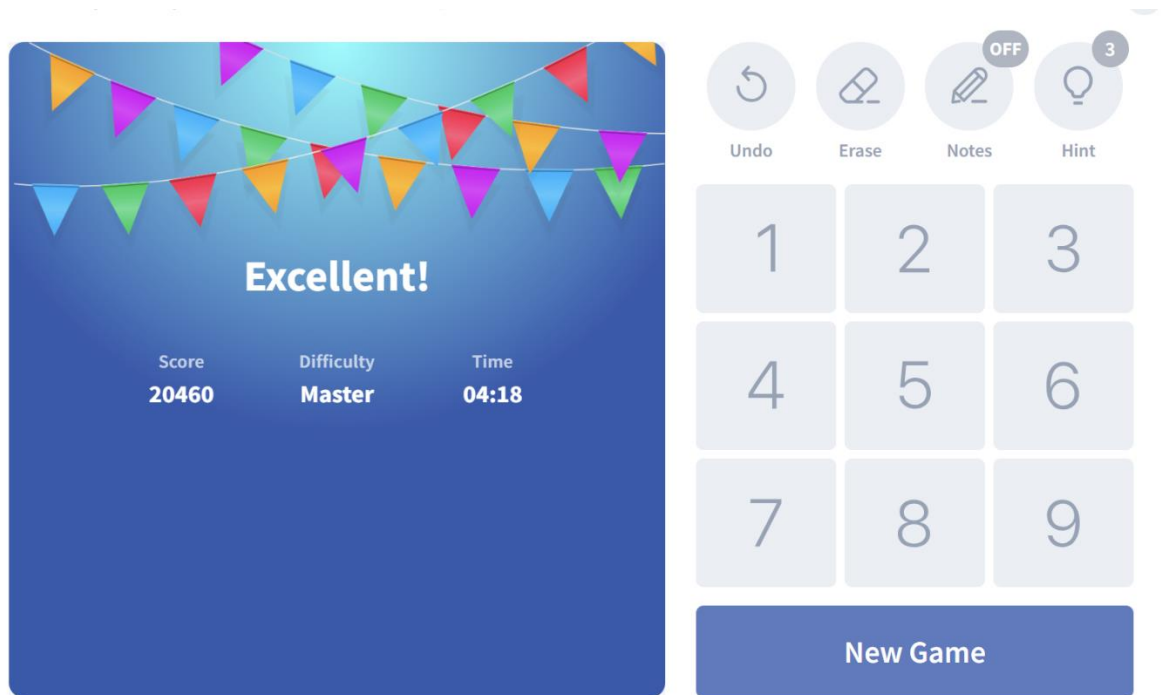
```
chirag-21@chirag-21-VirtualBox:~$ vi sudokupuzzle.pl
chirag-21@chirag-21-VirtualBox:~$ perl sudokupuzzle.pl
Enter the Sudoku puzzle row by row. Use '0' for empty cells.
Enter row 0: 9 0 0 0 0 0 0 0 4
Enter row 1: 0 3 0 0 7 0 6 9 0
Enter row 2: 0 0 2 8 0 0 0 0 0
Enter row 3: 0 5 0 0 0 0 0 0 1
Enter row 4: 0 0 0 0 4 0 0 0 3
Enter row 5: 8 0 0 7 0 0 4 5 0
Enter row 6: 3 0 0 0 0 9 0 0 0
Enter row 7: 0 0 0 0 0 0 0 1 0
Enter row 8: 0 9 0 0 6 0 5 7 0
```

Solved Sudoku:

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

```
chirag-21@chirag-21-VirtualBox:~$
```





OBSERVATION:-

There have been some critical findings observed while accomplishing this assignment which are as follows→

- The significant uses of 'split `\s+/, $input`' & '`$input`' which represent a function and a variable respectively and their important utilization could be described as;
The split function in Perl is used to split a string into a list of substrings based on a specified delimiter pattern.
In this case, the delimiter pattern is specified as `\s+/,`. Here, `\s+/,` is a regular expression that matches one or more whitespace characters (`\s` represents whitespace, and `+` indicates one or more occurrences). Therefore, the line `split \s+/, $input;` splits the user input string (`$input`) into a list of values wherever one or more whitespace characters are encountered.

- This variable holds the user input for a single row of the Sudoku puzzle. It is obtained using the `<STDIN>` operator, which reads a line from the standard input (usually the keyboard) and assigns it to the variable.

- Also, another important observation is that, user while entering the input sudoku quiz must reckon in his/her mind that input is taken row wise and must be entered with whitespaces in between and only values between `1-9` (in single digit) will be considered valid, along with `0` wherever empty cells are present.

Below mentioned are some real world applications our special assignment→

- Entertainment and Recreation:
Sudoku puzzle solvers can be integrated into gaming apps, websites, or printed puzzle books.
- Learning and Skill Development:
Educational platforms can use Sudoku solvers as a teaching tool to help students understand problem-solving algorithms. Solvers provide step-by-step solutions, aiding in the development of logical reasoning and critical thinking skills.
- AI and Machine Learning Training:
Sudoku puzzles are often used as benchmarks to test and train the capabilities of AI models.

CONCLUSION:-

- Our developed sudoku puzzle solver game solved the master level sudoku puzzle from “sudoku.com” in just 1min.58sec. accurately with 0 mistakes whose result has been published in the output section could be sighted as the major concluding part of this report.
- The optimum and efficient use of three subroutines (in_row, in_col, in_box) which are helper functions used by is_valid_move. They check whether a number is already present in a specific row, column, or 3x3 subgrid, respectively could be concluded as a vital part of the program.

- This project could serve as an example of applying programming and algorithmic concepts to solve a real-world problem.
Further enhancements could include additional features such as user-friendly interfaces, puzzle generation, and improved error handling.
Overall, the project demonstrates the power and flexibility of Perl for solving logical puzzles.

Citations & References →

- **www.lucidchart.com/flowcharts**
- **sudoku.com**
- **<https://www.geeksforgeeks.org/building-and-visualizing-sudoku-game-using-pygame/>**
- **<https://www.geeksforgeeks.org/backtracking-algorithms/>**