|  |  |
| --- | --- |
| **SUDOKU**  **SOLVER** |  |

**Submitted To:**

**Name: Ashifatul Ferdousi**

**Designation: Assistant Lecturer**

**Department: CSE**

**Institution: IIUC**

**Email: ashifafulferdousichaity70@gmail.com**

**Submitted By:**

**Group No: 11**

**ID 1: C223204 Name: Bibi Ayesha Akhter Omera**

**Email: c223204@ugrad.iiuc.ac.bd**

**ID 2: C223210 Name: Fahmida Yasmin**

**Email: c223210@ugrad.iiuc.ac.bd**

**Section: 4AF**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Serial Number** | **Contents** | **Page Number** |
| 01 | Abstract | 01 |
| 02 | Project Overview | 01-02 |
| 03 | Objectives | 02 |
| 04 | Features | 02 |
| 05 | Development Environment | 03 |
| 06 | Methodology | 03-04 |
| 07 | Code | 05 |
| 09 | Output of the Code | 05 |
| 10 | Time & Space Complexity | 06 |
| 11 | Future Enhancements | 07 |
| 12 | Conclusion | 07 |
| 13 | Reference | 08 |

**Abstract**

The Sudoku Solver project is a web-based application designed to both generate and solve Sudoku puzzles, providing an engaging and educational tool for puzzle enthusiasts. Sudoku, a widely popular number puzzle, requires filling a 9x9 grid such that each row, column, and 3x3 sub-grid contains all the numbers from 1 to 9 without repetition. This project leverages modern web technologies, including HTML, CSS, and JavaScript, to create an interactive interface where users can fetch randomly generated puzzles and have them solved algorithmically. The core of the solver is a backtracking algorithm, a well-known method for solving constraint satisfaction problems like Sudoku. By combining user-friendly design with robust algorithmic implementation, the Sudoku Solver project aims to enhance both the user experience and the educational value of the puzzle-solving process.

The application fetches Sudoku puzzles from an external API, allowing users to select puzzles of varying difficulty levels. Once a puzzle is fetched, users can either attempt to solve it themselves or use the application's solving feature to see the solution. The solver ensures that all Sudoku rules are strictly followed, providing valid solutions for any given puzzle. This project not only demonstrates the practical application of web development skills but also serves as a showcase for implementing algorithms in a real-world scenario.

**Project Overview**

This project addresses the essential conditions required to solve a Sudoku puzzle while offering a seamless user experience through its web-based interface. The core of the project is centered around solving the puzzle by ensuring three fundamental constraints are met:

**>>** **Row Constraint:** Each number must appear only once in each row.

**>>** **Column Constraint:** Each number must appear only once in each column.

**>> Sub-grid Constraint:** Each number must appear only once in each of the nine 3x3 sub-grids.

The project utilizes these constraints to develop a reliable solver using a backtracking algorithm, which systematically fills the grid and backtracks when it encounters an invalid state. If a number can be placed without violating constraints, the algorithm proceeds to the next cell and attempts to place another number. This process continues recursively, advancing to the next cell each time a valid placement is made. If the algorithm reaches a cell where no valid number can be placed, it backtracks to the previous cell and tries the next possible number. This step is crucial as it helps in discarding paths that do not lead to a solution and exploring alternative paths. The process continues until the algorithm successfully fills all cells of the grid with valid numbers, leading to a solved puzzle. If the algorithm exhausts all possibilities and cannot fill the grid, it concludes that the puzzle is unsolvable.

**Objectives**

**>> Develop an Interactive Sudoku Solver :** Create a user-friendly web interface where users can fetch and solve Sudoku puzzles.

**>> Implement a Robust Solving Algorithm :** Use a backtracking algorithm to ensure accurate and efficient puzzle solving.

**>> Ensure Cross-Browser Compatibility :** Design the application to work seamlessly across different web browsers.

**>> Provide Visual Feedback :**  Implement dynamic visual feedback to highlight changes and provide an engaging user experience.

**Features**

**>> Puzzle Generation :** Fetch Sudoku puzzles from an external API.

**>> Puzzle Solving :**  Automatically solve fetched puzzles using a backtracking algorithm.

**>> Dynamic User Interface :** Interactive grid with visual feedback for filled cells.

**>> Difficulty Levels :** Option to fetch puzzles of varying difficulty (easy, medium, hard).

**>> Error Handling :** Manage and display errors gracefully when fetching or solving puzzles.

**Development Environment**

**Software Requirements:**

**Web Browser:** Any modern web browser (Chrome, Firefox, Safari, Edge)

**Code Editor:** Visual Studio Code, Sublime Text, or any preferred editor

**Libraries:** None required beyond standard JavaScript

**Operating System:**  Windows, macOS, or Linux

**Tools:** Node.js (for potential server-side development), Git (for version control)

**Hardware Requirements:**

**Processor:**  Intel Core i3 or equivalent

**RAM:** 4GB minimum

**Storage:** 50MB for project files and dependencies

**Methodology**

The methodology employed in the Sudoku Solver project focuses on integrating web technologies and algorithmic solutions to create a robust platform for solving Sudoku puzzles interactively. By leveraging HTML, CSS, and JavaScript, the project establishes a user-friendly interface that facilitates puzzle fetching, display, and real-time solving capabilities.

**>> Fetching Puzzles:**

**API Integration:** The project utilizes the Sugoku API to fetch Sudoku puzzles. When the user clicks the "GetPuzzle" button, an API request is made to fetch a puzzle of the chosen difficulty level (easy, medium, or hard).

**Data Handling:** The API response contains a Sudoku board in JSON format. The fetched board is stored in a JavaScript array, board, representing the 9x9 grid. The FillBoard(board) function is then called to display the fetched puzzle on the web page by populating the grid cells with the numbers from the API response.

**>> Solving Puzzles:**

**User Interaction:** The user can click the "SolvePuzzle" button to initiate the solving process.This action triggers the SudokuSolver(board, 0, 0, 9) function, starting the backtracking algorithm to solve the puzzle.

**>> Backtracking Algorithm:**

**Initialization:** The algorithm begins at the first cell (0,0) and attempts to place numbers from 1 to 9.

**Validation:** For each number, the isValid(board, row, col, num, n) function checks if placing the number violates any Sudoku rules (row, column, or sub-grid constraints).

**Recursion:** If a valid number is found, it is placed in the cell, and the algorithm recursively moves to the next cell.

**Backtracking:** If no valid number can be placed in a cell, the algorithm backtracks to the previous cell, removes the last placed number, and tries the next possible number.

**Completion:** The algorithm continues this process until the entire grid is filled with valid numbers, resulting in a solved puzzle.

By integrating the Sugoku API for puzzle generation and employing a backtracking algorithm for puzzle solving, the Sudoku Solver project demonstrates the practical application of web technologies and algorithmic problem-solving in a cohesive and interactive manner.

**Code**

**Main Sudoku solver code**

**#include<iostream>**

**using namespace std;**

**void Print(int board[][9], int n)**

**{**

**for (int i = 0; i < n; i++)**

**{**

**for (int j = 0; j < n; j++)**

**{**

**cout << board[i][j] << " ";**

**}**

**cout << endl;**

**}**

**cout << endl;**

**}**

**bool isValid(int board[][9], int row, int col, int num, int n)**

**{**

**for (int i = 0; i < n; i++)**

**{**

**if (board[i][col] == num)**

**return false;**

**if (board[row][i] == num)**

**return false;**

**if (board[3 \* (row / 3) + i / 3][3 \* (col / 3) + i % 3] == num)**

**return false;**

**}**

**return true;**

**}**

**bool SudokuSolver(int board[][9], int i, int j, int n)**

**{**

**if (i == n)**

**{**

**Print(board, n);**

**return true;**

**}**

**if (j == n)**

**{**

**return SudokuSolver(board, i + 1, 0, n);**

**}**

**if (board[i][j] != 0)**

**{**

**return SudokuSolver(board, i, j + 1, n);**

**}**

**for (int num = 1; num <= 9; num++)**

**{**

**if (isValid(board, i, j, num, n))**

**{**

**board[i][j] = num;**

**if(SudokuSolver(board, i, j + 1, n) == true)**

**return true;**

**else**

**board[i][j] = 0;**

**}**

**}**

**return false;**

**}**

**int main()**

**{**

**int n = 9;**

**int board[9][9] =**

**{**

**{0, 0, 7, 1, 0, 0, 0, 6, 0},**

**{1, 0, 5, 2, 0, 8, 0, 0, 0},**

**{6, 0, 0, 0, 0, 7, 1, 2, 0},**

**{3, 1, 2, 4, 0, 5, 0, 0, 8},**

**{0, 0, 6, 8, 9, 0, 2, 0, 0},**

**{0, 0, 0, 0, 0, 3, 0, 0, 1},**

**{0, 0, 1, 0, 0, 4, 9, 8, 6},**

**{0, 8, 3, 9, 0, 6, 0, 0, 0},**

**{0, 6, 0, 0, 8, 2, 7, 0, 3},**

**};**

**SudokuSolver(board, 0, 0, n);**

**return 0;**

**}**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Sudoku Solver</title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <div class="Heading">Sudoku Solver</div>

    <div class="sudoku\_grid">

        <div id="0" class="box1 bl bt"></div>

        <div id="1" class="box1 bt"></div>

        <div id="2" class="box1 br bt"></div>

        <div id="3" class="box2 bt"></div>

        <div id="4" class="box2 bt"></div>

        <div id="5" class="box2 br bt"></div>

        <div id="6" class="box3 bt"></div>

        <div id="7" class="box3 bt"></div>

        <div id="8" class="box3 br bt"></div>

        <div id="9" class="box1 bl"></div>

        <div id="10" class="box1"></div>

        <div id="11" class="box1 br"></div>

        <div id="12" class="box2"></div>

        <div id="13" class="box2"></div>

        <div id="14" class="box2 br"></div>

        <div id="15" class="box3"></div>

        <div id="16" class="box3"></div>

        <div id="17" class="box3 br"></div>

        <div id="18" class="box1 bb bl"></div>

        <div id="19" class="box1 bb"></div>

        <div id="20" class="box1 br bb"></div>

        <div id="21" class="box2 bb"></div>

        <div id="22" class="box2 bb"></div>

        <div id="23" class="box2 br bb"></div>

        <div id="24" class="box3 bb"></div>

        <div id="25" class="box3 bb"></div>

        <div id="26" class="box3 bb br"></div>

        <div id="27" class="box1 bl"></div>

        <div id="28" class="box1"></div>

        <div id="29" class="box1 br"></div>

        <div id="30" class="box2"></div>

        <div id="31" class="box2"></div>

        <div id="32" class="box2 br"></div>

        <div id="33" class="box3"></div>

        <div id="34" class="box3"></div>

        <div id="35" class="box3 br"></div>

        <div id="36" class="box1 bl"></div>

        <div id="37" class="box1"></div>

        <div id="38" class="box1 br"></div>

        <div id="39" class="box2"></div>

        <div id="40" class="box2"></div>

        <div id="41" class="box2 br"></div>

        <div id="42" class="box3"></div>

        <div id="43" class="box3"></div>

        <div id="44" class="box3 br"></div>

        <div id="45" class="box1 bb bl"></div>

        <div id="46" class="box1 bb"></div>

        <div id="47" class="box1 br bb"></div>

        <div id="48" class="box2 bb"></div>

        <div id="49" class="box2 bb"></div>

        <div id="50" class="box2 br bb"></div>

        <div id="51" class="box3 bb"></div>

        <div id="52" class="box3 bb"></div>

        <div id="53" class="box3 bb br"></div>

        <div id="54" class="box1 bl"></div>

        <div id="55" class="box1"></div>

        <div id="56" class="box1 br"></div>

        <div id="57" class="box2"></div>

        <div id="58" class="box2"></div>

        <div id="59" class="box2 br"></div>

        <div id="60" class="box3"></div>

        <div id="61" class="box3"></div>

        <div id="62" class="box3 br"></div>

        <div id="63" class="box1 bl"></div>

        <div id="64" class="box1"></div>

        <div id="65" class="box1 br"></div>

        <div id="66" class="box2"></div>

        <div id="67" class="box2"></div>

        <div id="68" class="box2 br"></div>

        <div id="69" class="box3"></div>

        <div id="70" class="box3"></div>

        <div id="71" class="box3 br"></div>

        <div id="72" class="box1 bb bl"></div>

        <div id="73" class="box1 bb"></div>

        <div id="74" class="box1 br bb"></div>

        <div id="75" class="box2 bb"></div>

        <div id="76" class="box2 bb"></div>

        <div id="77" class="box2 br bb"></div>

        <div id="78" class="box3 bb"></div>

        <div id="79" class="box3 bb"></div>

        <div id="80" class="box3 bb br"></div>

    </div>

    <div class="Buttons">

        <button id="GetPuzzle"> GetPuzzle </button>

        <button id="SolvePuzzle"> SolvePuzzle </button>

    </div>

    <script src="./Script.js"></script>

</body>

</html>

**Style.css**

#name{

    width: 60%;

    display: flex;

    align-items: center;

    font-size: 50px;

    font-weight: bold;

}

.Heading{

    width: 100%;

    display: flex;

    justify-content: center;

    align-items: center;

    font-size: 40px;

    font-weight: bolder;

}

.sudoku\_grid{

    display: flex;

    flex-direction: row;

    max-width: 578px;

    flex-wrap: wrap;

    margin: 10px auto ;

}

.sudoku\_grid div{

    display:  flex;

    align-items: center;

    justify-content: center;

    width: 60px;

    height: 60px;

    color: white;

    font-size: 40px;

    box-sizing: border-box;

}

.Buttons{

    display: flex;

    justify-content: center;

    width: 100%;

    margin-bottom: 80px;

}

#GetPuzzle,#SolvePuzzle{

    display: inline-block;

    text-decoration: none;

    color: #fff;

    font-weight: bold;

    background-color: #538fbe;

    padding: 10px 35px;

    font-size: 24px;

    border: 1px solid #2d6898;

    box-shadow: inset 0px 1px 0px rgba(255,255,255,.3);

    margin-right: 10px;

}

.box1{

    background-color: #1520A6;

    border: 2px solid  white;

}

.box1:hover{

    background-color: #48aaaD;

    color: black;

}

.box2{

    background-color: #BC5448;

    border: 2px solid  white;

}

.box2:hover{

    background-color: red;

}

.box3{

    background-color: #028a0f;

    border: 2px solid  white;

}

.box3:hover{

    background-color: #3Ded97;

    color: black

}

.bl{

    border-left: 4px solid black;

}

.bb{

    border-bottom: 4px solid black;

}

.br{

    border-right: 4px solid black;

}

.bt{

    border-top: 4px solid black;

}

**Script.js**

var arr = Array.from({ length: 9 }, () => Array(9).fill(null));

// Populate the arr array with references to the div elements

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

    for (var j = 0; j < 9; j++) {

        arr[i][j] = document.getElementById(i \* 9 + j);

    }

}

var board = Array.from({ length: 9 }, () => Array(9).fill(0));

// Function to fill the board on the webpage

function FillBoard(board) {

    for (var i = 0; i < 9; i++) {

        for (var j = 0; j < 9; j++) {

            if (board[i][j] != 0) {

                arr[i][j].innerText = board[i][j];

            } else {

                arr[i][j].innerText = '';

            }

        }

    }

}

// Get the buttons for fetching and solving the puzzle

let GetPuzzle = document.getElementById('GetPuzzle');

let SolvePuzzle = document.getElementById('SolvePuzzle');

// Fetch a puzzle from the `sugoku` API

GetPuzzle.onclick = function () {

    fetch('https://sugoku.onrender.com/board?difficulty=easy')

        .then(response => response.json())

        .then(data => {

            console.log(data);

            board = data.board;

            FillBoard(board);

        })

        .catch(error => console.error('Error fetching puzzle:', error));

};

// Solve the puzzle when the SolvePuzzle button is clicked

SolvePuzzle.onclick = () => {

    SudokuSolver(board, 0, 0, 9);

};

// Function to check if a number can be placed in a cell

function isValid(board, row, col, num, n) {

    for (let i = 0; i < n; i++) {

        if (board[i][col] == num) return false;

        if (board[row][i] == num) return false;

        if (board[Math.floor(row / 3) \* 3 + Math.floor(i / 3)][Math.floor(col / 3) \* 3 + i % 3] == num) return false;

    }

    return true;

}

// Function to solve the Sudoku puzzle using backtracking

function SudokuSolver(board, i, j, n) {

    if (i == n) {

        FillBoard(board);

        return true;

    }

    if (j == n) {

        return SudokuSolver(board, i + 1, 0, n);

    }

    if (board[i][j] != 0) {

        return SudokuSolver(board, i, j + 1, n);

    }

    for (let num = 1; num <= 9; num++) {

        if (isValid(board, i, j, num, n)) {

            board[i][j] = num;

            if (SudokuSolver(board, i, j + 1, n)) return true;

            board[i][j] = 0;

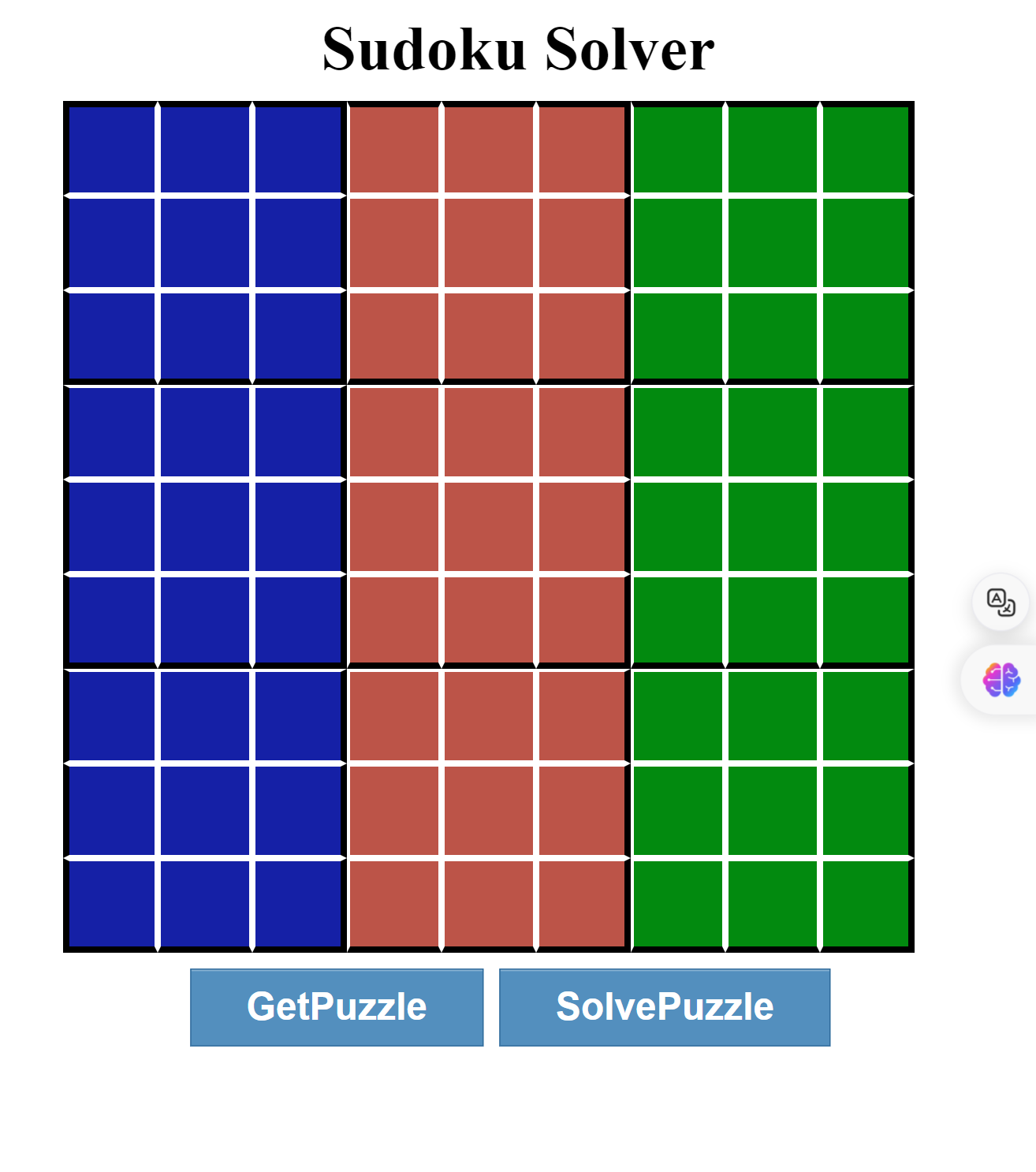
        }

    }

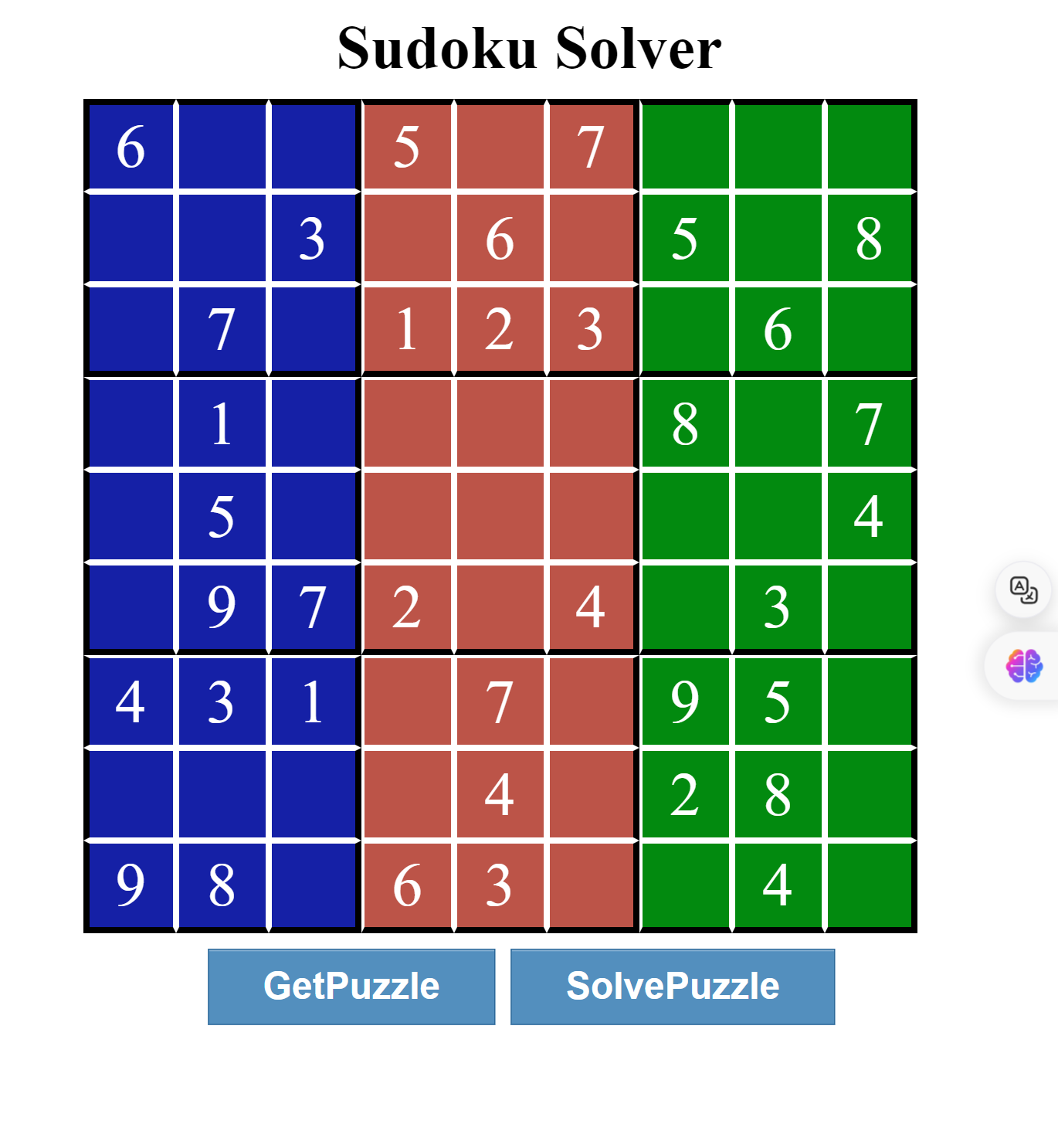
    return false;

}

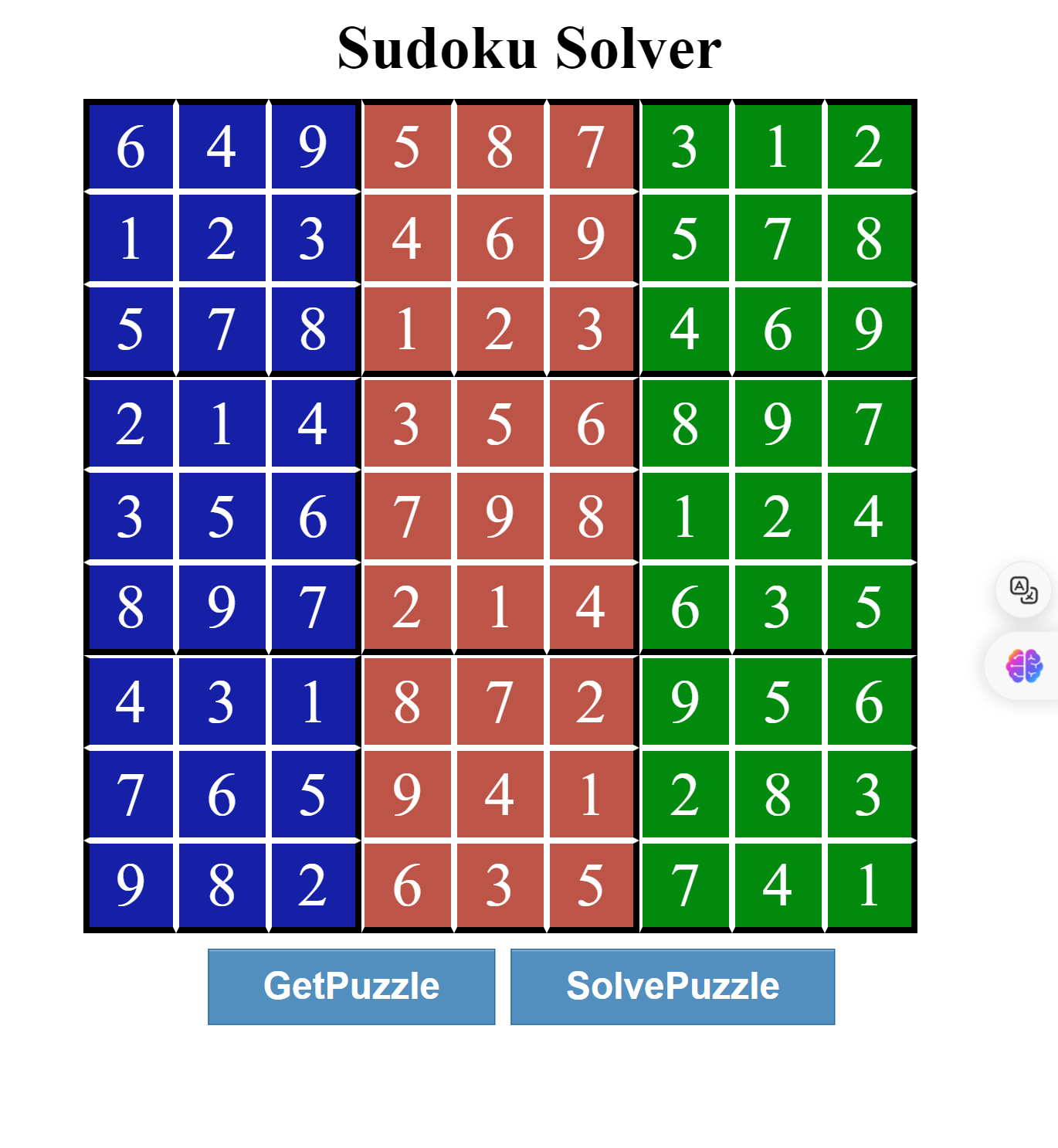
**Output of the Code**



After Enter the GetPuzzle Button



After Enter the SolvePuzzle Button



Time and Space Complexity

**Time Complexity:**

The time complexity of the Sudoku Solver project primarily revolves around the backtracking algorithm used to solve the Sudoku puzzles. The backtracking algorithm employed for solving Sudoku puzzles typically operates with a time complexity of approximately (O(9^(n\*n))), where (n) is the size of the Sudoku grid (in this case, (n = 9)). This complexity arises because for each empty cell, the algorithm potentially tries out 9 different numbers (1 through 9), leading to a branching factor of 9 for each recursive call. The worst-case scenario occurs when the algorithm needs to backtrack multiple times, exploring all possible configurations until a solution is found or determined to be impossible.

**Total Time Complexity: O(9^(n\*n))**

**Space Complexity:**

The space complexity of the Sudoku Solver project can be broken down as follows:

**>> Grid Storage:** The Sudoku grid itself is stored in two main arrays:

**- “board” :** This array holds the initial puzzle fetched from the API and serves as the working grid for the solver. Its space complexity is (O(n^2)), where (n = 9) (for a 9x9 grid).

**- “arr” :** An array of references to HTML elements representing the grid cells on the webpage. This array does not contribute significantly to the overall space complexity as it only stores references.

**>> Function Call Stack** : During the execution of the backtracking algorithm, recursive function calls are made to explore different possibilities for filling the grid. The maximum depth of the recursion stack is bounded by the number of empty cells in the grid, which in the worst case is (O(n^2)). Therefore, the space complexity due to recursion is (O(n^2)) .

**>> Auxiliary Space:** The algorithm also uses additional space for temporary variables and parameters passed during function calls, but these are constant factors and do not contribute significantly to the overall space complexity in Big O notation.

**Total Space Complexity: O(n^2)**

These complexities indicate that while the solver efficiently handles standard 9x9 Sudoku puzzles, it may experience performance challenges with larger grids or highly complex puzzles due to the exponential nature of the backtracking algorithm.

**Future Enhancements**

**>> Enhanced UI/UX:** Add animations and improve the interface design for better user engagement.

**>> User Input:**  Allow users to input their own puzzles for solving.

**>> Hint System:** Implement a feature to provide hints to users without solving the entire puzzle.

**>> Mobile Optimization:** Ensure the application is fully responsive and optimized for mobile devices.

**>> Difficulty Selection**: Allow users to select the difficulty level of the puzzle directly from the interface.

**>> Optimized Performance for Larger Puzzles:** Enhance the solver's efficiency to handle larger Sudoku grids beyond the standard 9x9 size, ensuring smooth and responsive solving even with increased complexity.

**Conclusion**

The Sudoku Solver project not only showcases the effective application of web technologies but also highlights the intersection of algorithmic problem-solving and recreational gaming. Through seamless integration with an external API for puzzle generation and a meticulous implementation of the backtracking algorithm, the project offers a versatile tool suitable for Sudoku enthusiasts of all levels.

The interactive interface and real-time visual feedback not only facilitate puzzle-solving but also contribute to a rich user experience. This combination of functionality and usability makes the Sudoku Solver both educational and entertaining, providing users with a dynamic platform to engage with and learn from Sudoku puzzles.

**Reference**

**>> GeeksforGeeks**

[**https://www.geeksforgeeks.org/sudoku-backtracking-7/**](https://www.geeksforgeeks.org/sudoku-backtracking-7/)

**>> Sugoku API**

[**https://github.com/bertoort/sugoku**](https://github.com/bertoort/sugoku)

**>> Mozilla Developer Network (MDN) Web Docs**

[**https://developer.mozilla.org/en-US/docs/Web/CSS**](https://developer.mozilla.org/en-US/docs/Web/CSS)

**>>Javatpoint**

[**https://www.javatpoint.com/backtracking-introduction**](https://www.javatpoint.com/backtracking-introduction)

**>> Stack Overflow**

**https://stackoverflow.com/**