

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240070049 |
| **NAME** | DEEPAK G |
| **PROJECT TITLE** | INTERACTIVE SUDOKU SOLVER |
| **DATE OF SUBMISSION** | 29-10-2025 |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

Artificial Intelligence (AI) is a branch of computer science that enables machines to make decisions, learn, and solve problems like humans. It plays a vital role in simplifying human tasks by providing intelligent automation.

This mini-project, **“Interactive Sudoku Solver”**, demonstrates the use of a powerful AI search technique called **Backtracking** to solve the classic 9x9 Sudoku puzzle. The algorithm systematically explores possible number combinations, eliminating invalid paths to find the unique solution. This simple application highlights how AI can be used to solve complex **Constraint Satisfaction Problems (CSPs)** efficiently.

**PROBLEM STATEMENT**

To develop an AI system that can **accurately and efficiently solve any valid Sudoku puzzle** provided by the user, and to visually demonstrate the step-by-step logic of the underlying **Backtracking search algorithm**.

**GOAL**

The main goal of this project is to build an AI agent capable of finding the solution to a Sudoku puzzle in a timely manner, even for hard difficulty levels, and to provide an interactive visualization of the solving process.

**Expected Results:**

* Accurately solve a standard 9x9 Sudoku puzzle by adhering to all rules.
* Apply the **Backtracking algorithm** for efficient depth-first search of the solution space.
* Provide a **visual representation** of the algorithm's process, including when it attempts a number and when it "backtracks" from a dead end.

**THEORETICAL BACKGROUND**

This project is based on **search algorithms**, a key component of Artificial Intelligence. The algorithm used here is **Backtracking**, which is a recursive, depth-first search technique.

### Literature Survey

* **Backtracking** is the most popular and systematic algorithm for solving Sudoku, as it is guaranteed to find all solutions (if they exist) by exploring the solution space.
* Other, simpler methods like a pure **Brute Force** search are less efficient because they do not "prune" invalid paths early, leading to an unnecessarily large search space.
* More advanced solvers often combine Backtracking with **Constraint Propagation** techniques to further reduce the search space and improve efficiency.

### Justification for Choosing Backtracking

* It is a **simple and classic AI search technique** ideal for solving Constraint Satisfaction Problems.
* It is **more efficient than pure Brute Force** because it *prunes* the search tree by immediately checking constraints and reversing a decision (backtracking) if a conflict is found.
* It is the **standard basis** for most high-performance Sudoku solvers, often combined with heuristics for optimization.

**ALGORITHM EXPLANATION WITH EXAMPLE**

The Backtracking algorithm works as a **recursive, depth-first search** following these steps (as described in the solver image):

1. **Find:** Scan the grid to find the **next empty cell** (a variable).
2. **Try:** Place a potential **valid number (1-9)** in that cell. A number is valid if it doesn't already exist in the same **row**, **column**, or $3 \times 3$ **sub-grid** (the constraints).
3. **Recurse:** If the number is valid, the algorithm recursively calls itself to solve the puzzle for the **next empty cell**.
4. **Backtrack:** If a recursive call reaches a point where no valid number can be placed in a cell (a dead end), it signifies a previous choice was wrong. It "**backtracks**" to the previous cell, erases the number there, and tries the next valid number.

This process continues until the entire grid is successfully filled (solution found) or it backtracks to the beginning having exhausted all possibilities (no solution exists).

**PSEUDOCODE**

## Backtracking Algorithm Pseudocode

The solver relies on two main functions: one to find the next empty cell, and the recursive function that attempts to fill that cell and then calls itself.

### 1. Find\_Empty\_Cell(grid)

This function scans the $9 \times 9$ Sudoku grid to locate the first cell that has a value of '0' (representing an empty cell).

Code snippet

function Find\_Empty\_Cell(grid):

FOR row from 0 to 8:

FOR col from 0 to 8:

IF grid[row][col] is 0:

RETURN (row, col)

RETURN NULL // No empty cells found, puzzle is solved

### 2. Is\_Valid(grid, row, col, num)

This function checks if placing a specific number (num) in a given cell (row, col) is legal according to Sudoku rules.

Code snippet

function Is\_Valid(grid, row, col, num):

// 1. Check Row Constraint

FOR c from 0 to 8:

IF grid[row][c] is num:

RETURN FALSE

// 2. Check Column Constraint

FOR r from 0 to 8:

IF grid[r][col] is num:

RETURN FALSE

// 3. Check 3x3 Subgrid (Box) Constraint

start\_row = row - (row MOD 3)

start\_col = col - (col MOD 3)

FOR r from start\_row to start\_row + 2:

FOR c from start\_col to start\_col + 2:

IF grid[r][c] is num:

RETURN FALSE

// If all checks pass

RETURN TRUE

### 3. Solve\_Sudoku(grid) (The Backtracking Core)

This is the main recursive function. It embodies the **Find**, **Try**, and **Backtrack** steps.

Code snippet

function Solve\_Sudoku(grid):

// 1. FIND: Find the next empty cell

pos = Find\_Empty\_Cell(grid)

IF pos is NULL:

RETURN TRUE // Base case: No empty cells, puzzle is solved

(row, col) = pos

// 2. TRY: Iterate through all possible numbers (1 to 9)

FOR num from 1 to 9:

IF Is\_Valid(grid, row, col, num) is TRUE:

// Place the number

grid[row][col] = num

// 3. RECURSE: Move to the next cell

IF Solve\_Sudoku(grid) is TRUE:

RETURN TRUE // A solution was found down this path

// 4. BACKTRACK: If the recursive call failed (returned FALSE),

// it means this number was wrong. Undo the choice and try the next number.

grid[row][col] = 0 // Erase the number, triggering the 'backtrack' visualization

RETURN FALSE // All numbers (1-9) failed for this cell; backtrack to the previous cell

**OUTPUT**

### **C:\Users\DEEPAK\AppData\Local\Packages\5319275A.WhatsAppDesktop_cv1g1gvanyjgm\TempState\FCCB3CDC9ACC14A6E70A12F74560C026\WhatsApp Image 2025-10-28 at 19.03.24_c555357c.jpg**

### Results

* The AI successfully solved the Sudoku puzzle by applying the **Backtracking** logic.
* The **"Visualize Solve"** feature effectively demonstrates the recursive nature and constraint-checking steps of the algorithm, showing how dead ends are avoided through backtracking.
* The use of backtracking, which involves **pruning** the search space, makes the solver highly efficient, especially compared to a brute force approach.

### Future Enhancements

* Implement a mechanism to **generate unique Sudoku puzzles** of varying difficulty levels.
* Incorporate more advanced **Constraint Propagation** heuristics (like Minimum Remaining Value (MRV) or Forward Checking) to optimize the backtracking process further.
* **Expand to other Sudoku variants** or similar Constraint Satisfaction Problems (CSPs).

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | [**https://github.com/DeepakG77/AI-MINI-PROJECT.git**](https://github.com/DeepakG77/AI-MINI-PROJECT.git) |

**REFERENCES**

 Russell, S. & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach (4th Edition)* – Pearson.

 GeeksforGeeks – *Backtracking Algorithm* – https://www.geeksforgeeks.org/backtracking-algorithm

 TutorialsPoint – *Artificial Intelligence Overview* – https://www.tutorialspoint.com/artificial\_intelligence

 Sudoku Solver: A Backtracking Approach - Kaggle – https://www.kaggle.com/code/mexwell/sudoku-solver-a-backtracking-approach