UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

DESIGN AND ANALYSIS OF ALGORITHM

PROJECT REPORT ON SUDOKU SOLVING GAME

SUBMITTED BY :-

RITINDER KAUR

UID : 24MCI10092

BRANCH: MCA(AIML)

SECTION/GROUP : 24MAM2A

SUBJECT CODE : 24CAP-612

SUBMITTED TO:

MS. DIVANYANSHI

E17438

ASSISTANT PROFESSOR

UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

1. Aim:

The aim of this code is to solve a Sudoku puzzle using a backtracking algorithm. It tries to fill empty cells (marked as 0) with numbers from 1 to 9 while following Sudoku rules. If a conflict arises, it backtracks to try other possibilities. If the puzzle is solvable, it prints the solution; otherwise, it indicates that no solution exists.

2. Objective:

The objective of this code is to solve a 9x9 Sudoku puzzle by filling in the empty cells (denoted by 0) while following the rules of Sudoku.

3. Problem definition:

1. Input:

A 9x9 Sudoku grid with some cells pre-filled, and others marked as 0 (empty).

2. Constraints:

- Each number from 1 to 9 must appear exactly once in:
- Each row
- Each column
- Each 3x3 subgrid

3. Goal:

- Find a valid assignment for all empty cells.
- Use backtracking to explore possible solutions systematically.
- If a solution exists, print the completed grid; if not, notify that no solution exists.

This problem demonstrates the use of recursion and backtracking to solve combinatorial puzzles efficiently.

4. Programming Languages used:

The code provided is written entirely in Python. It makes use of:

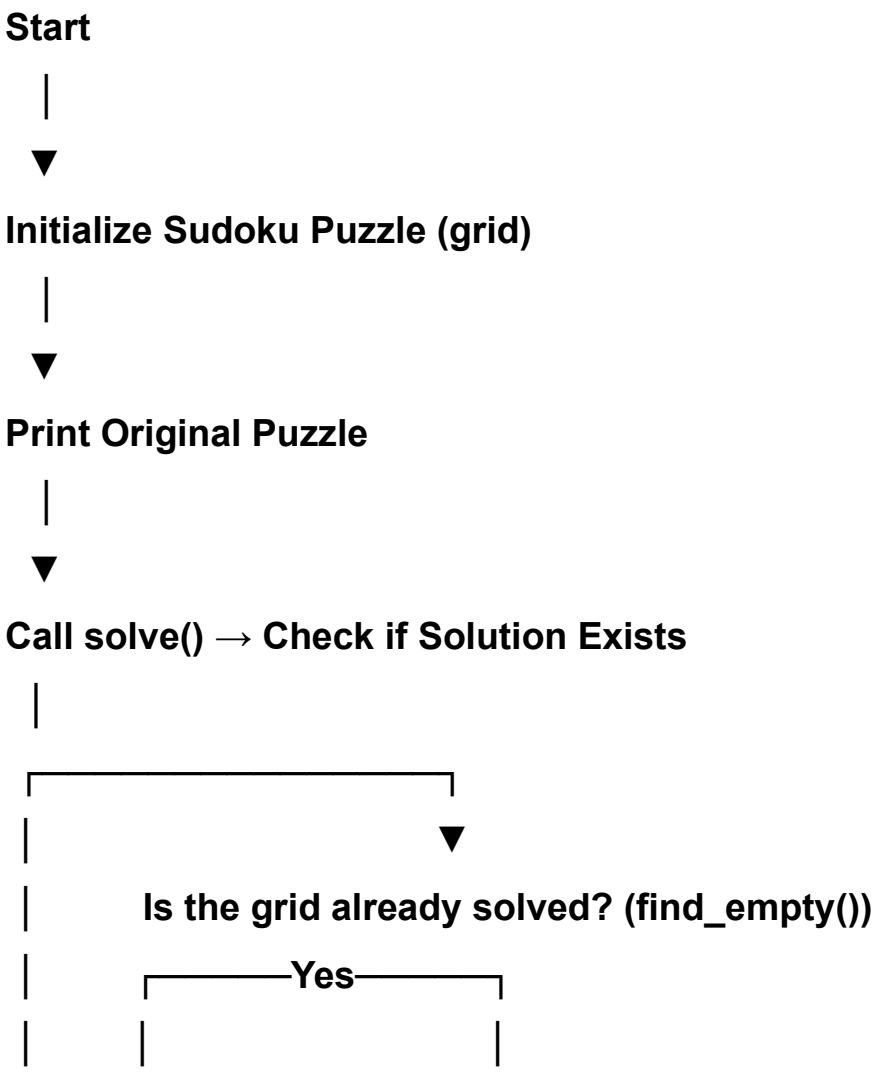
1. Python Core Language Features:

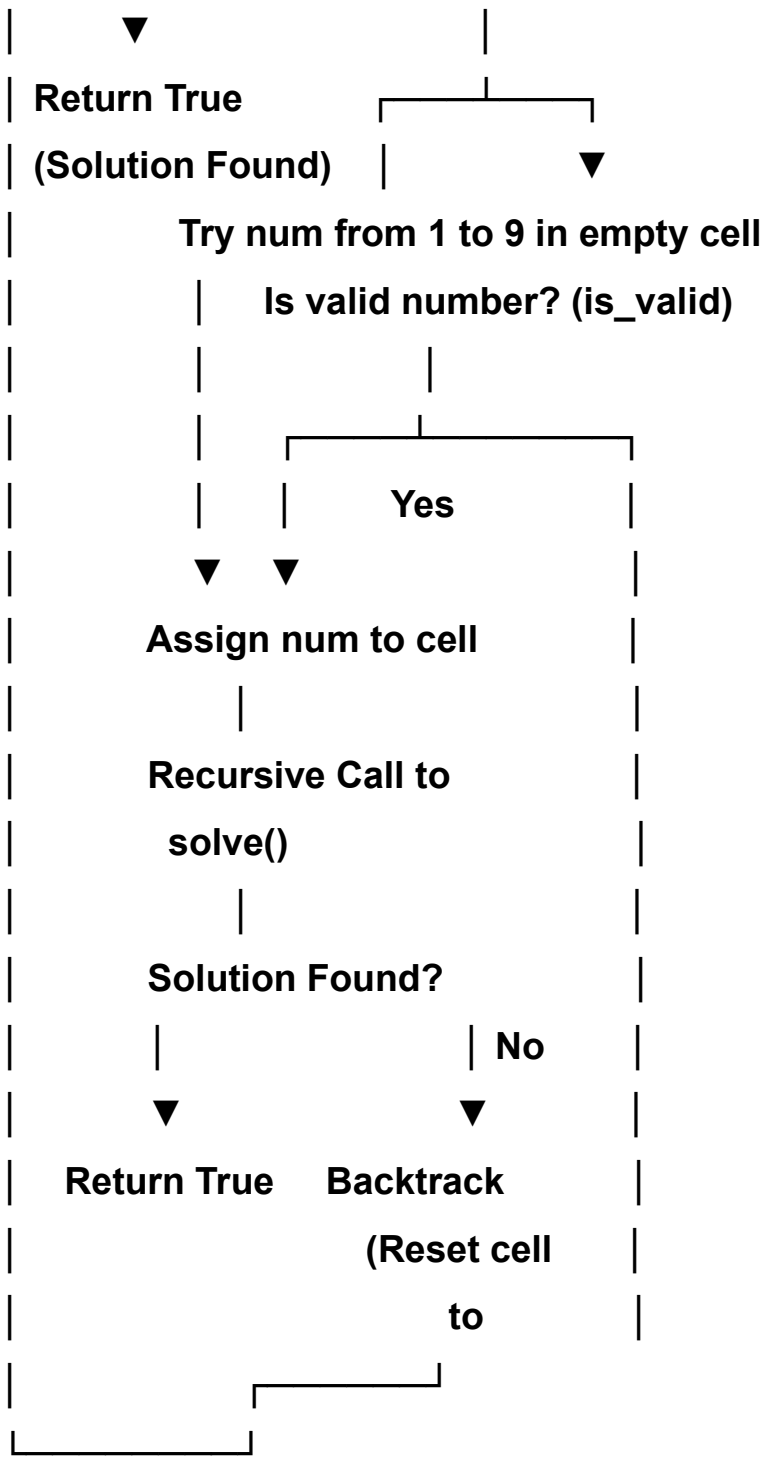
- **Classes and Objects**: The Sudoku class encapsulates the grid and methods for solving it.
- **Control Structures**: Uses for loops, if statements, and recursion in the solve() method.
- **Recursion**: The backtracking algorithm is implemented using recursive calls.

2. NumPy (Imported but Unused):

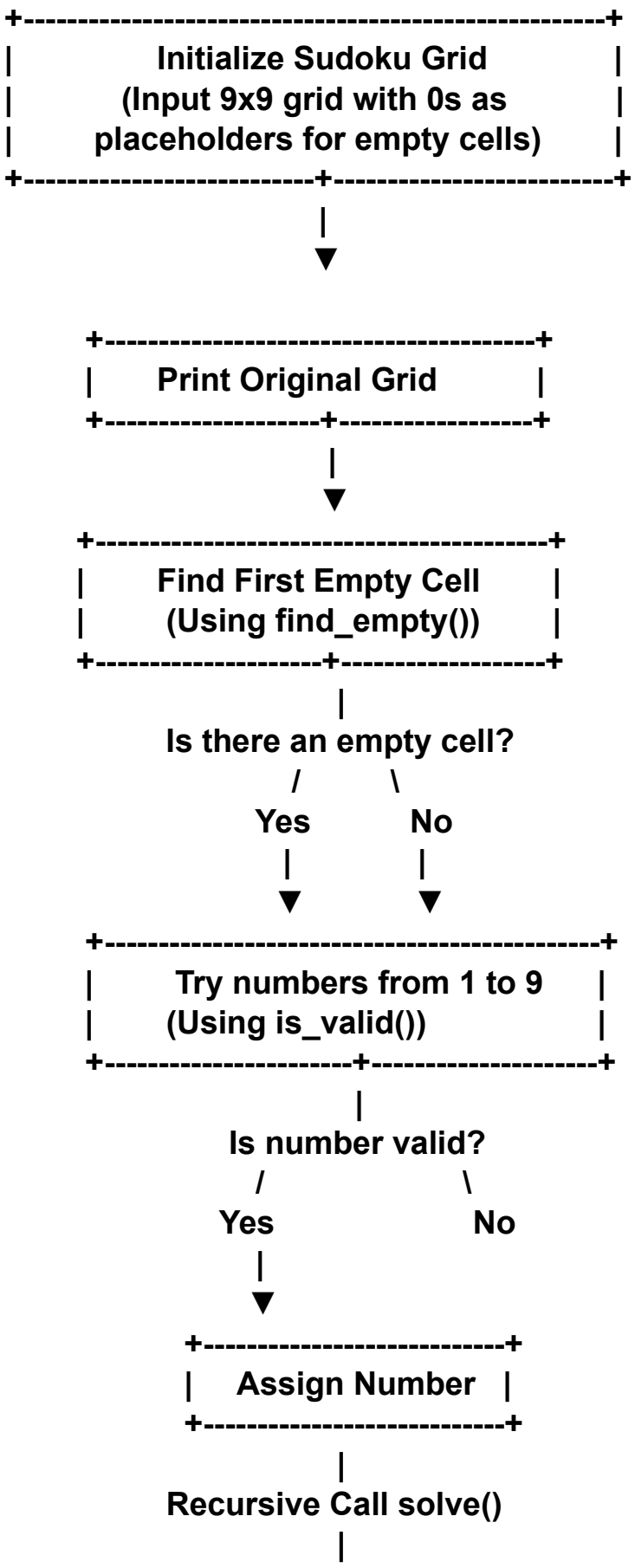
- Although NumPy is imported (import numpy as np), it is not used in this implementation. All operations are performed using native Python lists.

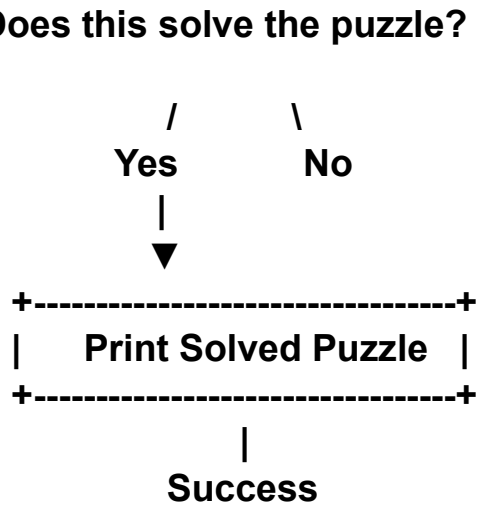
5. Flowchart:





6. **Block diagram:**


Initialize Sudoku Grid
(Input 9x9 grid with 0s as placeholders for empty cells)
Print Original Grid
Find First Empty Cell
(Using find_empty())
Is there an empty cell?
Yes
No
Try numbers from 1 to 9
(Using is_valid())
Is number valid?
Yes
No
Assign Number
Recursive Call solve()



7. Algorithm:

Step 1: Initialize the Sudoku Grid

- The Sudoku puzzle is represented as a 9x9 2D grid.
- 0s represent empty cells that need to be filled with valid numbers from 1 to 9.
- python

Eg.- sudoku_puzzle = [

[5, 3, 0, 0, 7, 0, 0, 0, 0],

[6, 0, 0, 1, 9, 5, 0, 0, 0],

[0, 9, 8, 0, 0, 0, 0, 6, 0],

]

Step 2: Create the Sudoku Object

- A Sudoku object is instantiated with the initial puzzle.

Eg.-sudoku = Sudoku(sudoku_puzzle)

Step 3: Print the Original Puzzle (Optional)

- The print_grid() method displays the current state of the grid.
- Empty cells (0s) are shown as

Eg-sudoku.print_grid()

Step 4: Solve the Puzzle Using Backtracking

- The solve() function is called to solve the puzzle recursively.

Step 5: Find the Next Empty Cell

- The find_empty() method scans the grid row by row to find the first empty cell (value 0).
- If no empty cells are found, the puzzle is solved.

Eg- empty = self.find_empty()

if not empty:

return True # Puzzle solved

Step 6: Try Numbers from 1 to 9

- For the current empty cell (row, col), try numbers from 1 to 9.

Step 7: Check if the Number is Valid

- The is_valid() function ensures that the selected number satisfies the Sudoku rules:
- Row Check: The number must not already exist in the same row.
- Column Check: The number must not already exist in the same column.
- 3x3 Sub-grid Check: The number must not already exist in the same 3x3 box.

Eg- if self.is_valid(row, col, num):

self.grid[row][col] = num

Step 8: Recursive Backtracking

- If the number is valid, place it in the empty cell.
- Recursively call solve() to try solving the next empty cells.

Eg-if self.solve():

return True

Step 9: Backtrack if No Valid Number Works

- If placing the current number does not lead to a solution, reset the cell to 0 (backtrack) and try the next number.

Eg- self.grid[row][col] = 0 # Backtrack

Step 10: Terminate When Solved or No Solution Exists

- If all numbers have been tried and no solution works, return False (backtrack further).
- If the entire grid is filled without empty cells, the puzzle is solved.

Eg- if sudoku.solve():

print("Solved Sudoku Puzzle:")

sudoku.print_grid()

else:

print("No solution exists.")

8. Implementation:

[1]:

```
import numpy as np

class Sudoku:
    def __init__(self, grid):
        self.grid = grid

    def print_grid(self):
        for row in self.grid:
            print(" ".join(str(num) if num != 0 else '.' for num in row))

    def is_valid(self, row, col, num):
        # Check row
        for i in range(9):
            if self.grid[row][i] == num:
                return False

        # Check column
        for i in range(9):
            if self.grid[i][col] == num:
                return False

        # Check 3x3 box
        box_row = row - row % 3
        box_col = col - col % 3
        for i in range(3):
            for j in range(3):
                if self.grid[box_row + i][box_col + j] == num:
                    return False

        return True
```

```
    def solve(self):
        empty = self.find_empty()
        if not empty:
            return True  # Solved

        row, col = empty

        for num in range(1, 10):
            if self.is_valid(row, col, num):
                self.grid[row][col] = num

                if self.solve():
                    return True

                # Backtrack
                self.grid[row][col] = 0

        return False

    def find_empty(self):
        for i in range(9):
            for j in range(9):
                if self.grid[i][j] == 0:
                    return (i, j)  # row, col
        return None
```

```
# Example Sudoku puzzle (0s represent empty cells)
sudoku_puzzle = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]

# Create Sudoku object
sudoku = Sudoku(sudoku_puzzle)

# Print the original puzzle
print("Original Sudoku Puzzle:")
sudoku.print_grid()

# Solve the puzzle
if sudoku.solve():
    print("\nSolved Sudoku Puzzle:")
    sudoku.print_grid()
else:
    print("No solution exists.")
```

9. Output:

```
Original Sudoku Puzzle:
5 3 . . 7 . . . .
6 . . 1 9 5 . . .
. 9 8 . . . . 6 .
8 . . . 6 . . . 3
4 . . 8 . 3 . . 1
7 . . . 2 . . . 6
. 6 . . . . 2 8 .
. . . 4 1 9 . . 5
. . . . 8 . . 7 9

Solved Sudoku Puzzle:
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
```

10. Conclusions:

1) Backtracking Algorithm:

The backtracking algorithm effectively solves Sudoku puzzles by exploring potential placements of numbers while adhering to Sudoku rules. This systematic approach allows for comprehensive searching through possibilities until a valid solution is found or all options are exhausted.

2) Validation Mechanism:

The implementation of a validation mechanism is crucial. The is_valid function ensures that each number placement adheres to Sudoku rules, preventing duplicate numbers in rows, columns, and 3x3 sub-grids. This significantly reduces the number of backtracking steps required, optimizing the solving process.

3) Handling Empty Cells:

The method for finding empty cells (find_empty) is essential for guiding the solver in identifying where to attempt number placements. It allows the algorithm to focus only on unfilled positions, streamlining the solving process.

4) Unique Solutions:

Most well-structured Sudoku puzzles are designed to have a unique solution, allowing for the backtracking approach to eventually yield a definitive answer. This is an important aspect of Sudoku puzzles, making them engaging and challenging.

5) Efficiency and Complexity:

While the algorithm has a theoretical worst-case complexity, in practice, it performs well due to the nature of typical Sudoku puzzles. The early rejection of invalid placements helps in navigating through fewer states, leading to faster solutions.

6) Educational Tool:

Implementing a Sudoku solver serves as an excellent exercise in programming and algorithm design. It reinforces concepts such as recursion, backtracking, and data structures while providing insights into problem-solving techniques applicable to a variety of domains.

7) Real World Applications:

The strategies used to solve Sudoku can be applied to other constraint satisfaction problems in fields such as operations research, scheduling, and artificial intelligence. The principles of validation, exploration, and backtracking are broadly applicable in many algorithmic contexts.

8) User Interaction:

The console-based interface demonstrates how simple user interaction can allow for displaying complex data structures (the Sudoku grid). The printed format makes it easy to visualize the puzzle's state before and after solving.

9) Potential Enhancements:

Future improvements could include features like difficulty level selection, hint systems, or graphical interfaces for better user engagement. Additionally, optimizing the search algorithm with heuristic techniques could enhance performance on more challenging puzzles.

11. Future frameworks:

The future of Sudoku-solving games can involve various frameworks and technologies that enhance gameplay, user experience, and educational value. Here are some potential future frameworks and developments that can be applied to Sudoku games:

1. Mobile Game Frameworks

- Unity: A powerful game engine for creating cross-platform games. It can provide high-quality graphics and sound, enhancing the gaming experience.
- Flutter: A UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase. It allows for responsive designs and smooth animations.

2. Web Development Frameworks

- React: A JavaScript library for building user interfaces. It can be used to create interactive web applications where users can play Sudoku online.
- Angular: A platform for building mobile and desktop web applications. It offers powerful tools for managing data flow and state, ideal for real-time Sudoku games.
- Vue.js: A progressive JavaScript framework for building user interfaces. It can be used for quick prototyping and developing interactive Sudoku games.

3. Game Engines

- Godot: An open-source game engine that's great for 2D games. It’s lightweight and has a user-friendly interface, making it suitable for creating Sudoku variants.
- Cocos2d: A framework for building 2D games. It’s particularly good for mobile game development and could be used to create engaging Sudoku experiences.

4. Educational Frameworks

- Kahoot!: A game-based learning platform that could integrate Sudoku puzzles for educational purposes, making learning logic and problem-solving fun.
- Scratch: A visual programming language that could be used to teach kids how to create their own Sudoku games, encouraging creativity and learning through coding.

5. Augmented Reality (AR) Frameworks

- ARKit / ARCore: Frameworks for building augmented reality experiences on iOS and Android. Future Sudoku games could use AR to overlay puzzles onto real-world surfaces, providing an interactive gaming experience.

- Unity with AR Foundation: Allows developers to create AR applications for multiple platforms. A Sudoku game could utilize AR to bring the puzzle into the physical space.

6. Artificial Intelligence Frameworks

- TensorFlow / PyTorch: These frameworks can be used to develop AI algorithms that generate Sudoku puzzles or provide hints based on user gameplay, making the game more dynamic and interactive.
- OpenAI Gym: Could be used to create reinforcement learning environments where AI learns to solve Sudoku puzzles, potentially providing insights into solving strategies.

7. Multiplayer Frameworks

- Photon: A framework for developing real-time multiplayer games. Future Sudoku games could include multiplayer modes where users compete against each other in real-time.
- Socket.io: A JavaScript library for real-time web applications, allowing for real-time interaction between players in a Sudoku game.

8. Community and Sharing Platforms

- Discord Bots: Create a Sudoku bot that users can interact with within a Discord server. This can include generating puzzles, solving them, or hosting tournaments.
- WebRTC: A technology that enables peer-to-peer communication in web browsers. Future Sudoku games could use this to enable players to solve puzzles together in real-time.

9. Gamification Frameworks

- PlayFab: A backend platform for building and managing live games, allowing developers to incorporate features like leaderboards, achievements, and user accounts.
- Gamify: Platforms that enable the addition of gamified elements to existing applications. Sudoku games could benefit from challenges, rewards, and progress tracking.

10. Cross-Platform Development

- Xamarin: A framework for building cross-platform mobile applications with a single codebase. This can help in reaching users on different devices seamlessly.
- Electron: A framework for building desktop applications using web technologies. Future Sudoku applications could be available on desktops while maintaining a consistent user experience across platforms.

12. Learning outcomes:

1. **Improved Problem-Solving Skills**

- Logical Thinking: Players must use logical reasoning to deduce the placement of numbers, enhancing their ability to analyze situations and make informed decisions.
- Strategic Planning: Sudoku requires planning several moves ahead, encouraging players to think strategically and anticipate potential challenges.

2. Enhanced Critical Thinking

- Pattern Recognition: Players develop the ability to recognize patterns and relationships between numbers, which is a valuable skill in mathematics and other fields.
- Analytical Skills: The need to evaluate the implications of each number placed on the board sharpens analytical thinking.

3. Cognitive Development

- Memory Improvement: Remembering rules and strategies can enhance working memory and recall abilities.
- Concentration and Focus: Successfully solving a Sudoku puzzle requires sustained attention, helping to improve concentration over time.

4. Mathematical Skills

- Number Sense: Players practice numerical skills and develop a better understanding of number properties and relationships.
- Basic Arithmetic: Although the focus is on logic rather than arithmetic, players reinforce their understanding of numbers as they place them correctly on the grid.

5. Persistence and Resilience

- Trial and Error: The iterative process of trying different solutions fosters resilience, teaching players that failure is part of the learning process.
- Overcoming Challenges: Completing difficult puzzles can build confidence and a sense of achievement, motivating players to tackle more challenging problems in the future.

6. Time Management

- Pacing: Players often set time limits for themselves, which can teach effective time management and the ability to work efficiently under pressure.
- Prioritization: Deciding which moves to make first requires prioritization skills, helping players learn to manage tasks effectively.

7. Social Skills

- Collaboration: Playing Sudoku with others can promote teamwork and communication skills, especially in a competitive or cooperative environment.
- Community Engagement: Engaging in Sudoku clubs or online forums fosters a sense of community and shared learning.

8. Stress Relief and Mental Health Benefits

- Relaxation: Solving puzzles can be a relaxing activity that reduces stress and anxiety, promoting mental well-being.
- Mindfulness: Focusing on the puzzle can serve as a form of mindfulness, helping players to be present in the moment.

9. Digital Literacy

- Technological Proficiency: Engaging with online Sudoku platforms or apps can enhance digital literacy, familiarizing players with technology and software applications.
- Understanding Game Mechanics: Learning how to navigate game interfaces and utilize digital tools can improve overall tech skills.

10. Creativity and Innovation

- Creative Problem Solving: While Sudoku is based on logic, players may develop innovative strategies for solving puzzles, fostering creative thinking.

13. Time complexity:

- **Worst Case:** O(9^(n)) where n is the number of empty cells. This is because, in the worst case, each empty cell might try all numbers from 1 to 9**.**