AI-Based Logic Puzzle Solver Using Constraint Satisfaction

CSE440-Section 1

Fatema Nazneen Zeba	1520381042
Nowshin Nower	2121008642
Mohammed Nafees Imtiaz	2212426642
Momtahina Quyyum	2211889042

OVERVIEW

Title: AI-Based Logic Puzzle Solver Using Constraint Satisfaction

Goal: Create an extensible, efficient logic puzzle solver (Sudoku, Kakuro)

Tools: Python, Tkinter, NumPy

Focus: Bridging CSP theory with interactive application

OBJECTIVES

- Build a flexible and scalable puzzle-solving system
- Implement constraint satisfaction algorithms
- Develop an interactive and intuitive GUI
- Support various puzzle types and dynamic feedback
- Emphasize maintainable and educational software design

Technical Stack

Language: Python 3.x

GUI Framework: Tkinter

Design Paradigms: OOP, Abstract Base Classes

Code Quality: Type hints, modular structure

Tools: Linters, virtual environments, unit tests

SYSTEM ARCHITECTURE

Modular, object-oriented design

Each puzzle encapsulated in its own class

Clear separation between logic, data, and GUI

Future puzzle integration via subclassing

CLASS HIERARCHY & DATA FLOW

LogicPuzzle: Abstract base for all puzzles

SudokuPuzzle: Grid-based, classic constraints

KakuroPuzzle: Sum-based logic with clues

LogicPuzzleGUI: Handles user interaction

Data Flow ⬇

CORE COMPONENTS

Logic Puzzle

Initializes grid, manages constraints

SudokuPuzzle

Handles row,column, subgrid validation

KakuroPuzzle

Supports clue validation and dynamic domains

Advanced: overlapping constraints, dynamic reconstruction

SOLVING ALGORITHMS

Backtracking: Classic DFS-based search

Constraint Propagation: Early pruning of invalid domains

Domain Reduction: Removes conflicting values from cells

Enhanced with optimizations and heuristics

GUI DESIGN

Elements: Grid, buttons, score display

Features: Live validation, hints, solution reveal

Accessibility: Keyboard shortcuts, color themes

User-friendly and educational layout

PERFORMANCE & OPTIMIZATION

- **Time Complexity**: Backtracking exponential, constraint checks linear
- **Space Complexity**: Grid $O(n^2)$, domain $O(1)$ per cell
- **Optimizations**: Forward checking, early termination, caching

FUTURE IMPROVEMENTS

Add more puzzles: Nonogram, KenKen, Crosswords

Use AI (e.g., reinforcement learning)

Multi-threaded solving

GUI upgrades: animations, themes, multi-language

Mobile version & cloud solving

REFERENCE

- Designed robust, scalable architecture
- Practical experience with CSP and GUI development
- Algorithm implementation with performance tuning
- Focus on HCI and user feedback
- Gained insights into AI, constraint logic, and OOP

Conclusion & References

- Project bridges CSP theory with real application
- Demonstrates educational and technical depth
- Ready for future scaling and AI integration

References include: Norvig, Dechter, Python/Tkinter docs,etc.

Thank You