Project 3: CSP to solve Sudoko

Omar Atef 7619
Ranim Mohareb 7771
Iyad Ashraf 7392

December 25, 2023

Abstract

Sudoku is a logic-based number-placement game that challenges players to fill a 9x9 grid with digits from 1 to 9. The objective is to complete the grid in such a way that each row, each column, and each of the nine 3x3 sub grids (also known as regions or boxes) contains all of the digits from 1 to 9 without repetition. The goal of this project is to provide a full game with GUI to show the AI agent solving the game and allow user to input board representation then agent solves it.

Contents

1 State Representation

- The simplest way to represent the board is using a 2D array. However, using 2D array uses ($9 \times 9 \times \texttt{sizeofinteger}$). This representation would limits the agent by only reaching short search depth and doesn't enable the agent to go deep in Finding the solution.
- Therefore we Represented the States with 1D array of int that makes the agent faster when solving the game.

2 Algorithms

2.1 Backtracking Algorithm

- The game supports Backtracking to be able to validate the input(to check that input puzzle is solvable) , Backtracking is also needed to generate random puzzle (fill random places of puzzle) to ensure that the puzzle generated is solvable

2.2 Puzzle Generation

Puzzle Generation includes the following key functions:

- 'fill_puzzle' Function:
  - Recursively fills a given Sudoku puzzle grid with valid values.
  - Utilizes functions like 'find_empty_cell', 'is_valid', and 'has_empty_cell'.
- 'solve_sudoko' Function:
  - Recursively solves a Sudoku puzzle using a backtracking approach.
  - Keeps track of the number of solutions using the global variable 'numberOfSolution'.
- 'generate_puzzle' Function:
  - Generates a Sudoku puzzle by first filling the puzzle grid completely and then selectively removing values to achieve the desired difficulty level.
  - Uses the 'fill_puzzle' and 'solve_sudoko' functions to ensure a valid and unique solution.

3 CSP

We defined the CSP as a class named 'CSP' (Constraint Satisfaction Problem) for solving Sudoku puzzles. Here is a summary of its key components:

- Initialization:

  - The constructor ‘__init__‘ initializes the CSP object with a given Sudoku board.
  - It sets up various data structures like ‘arcs‘, ‘variables‘, ‘heap‘, and ‘counts‘.

- ‘solve‘ Method:
  - Checks the consistency of the input Sudoku board using ‘check_arcs_consistency‘.
  - If the board is consistent, initiates the solving process using the ‘backtrack‘ method.
  - If a solution is found, updates the original board with the solution.

- ‘backtrack‘ Method:
  - Recursive backtracking algorithm for solving Sudoku.
  - Chooses the next variable to assign using a priority queue (‘heap‘).
  - Saves a copy of the domain for backtracking purposes.
  - Tries assigning values from the domain and recursively explores possible solutions.
  - Backtracks if a solution is not found.

- ‘create_variables‘ Method:
  - Initializes the variables for each cell in the Sudoku grid.
  - Populates domains based on the initial board values.
  - Sets up the priority queue (‘heap‘) for variable selection.

- ‘add_affected_arcs‘ and ‘add_arcs‘ Methods:
  - Adds arcs (constraints) for each variable based on its row, column, and box.
  - ‘add_affected_arcs‘ specifically adds arcs for a variable that has been assigned a value.

- ‘check_arcs_consistency‘ Method:
  - Checks the consistency of arcs (constraints) in the CSP.
  - Uses the ‘check_consistency‘ method to determine consistency between two variables.

- ‘check_consistency‘ Method:
  - Checks consistency between the domains of two variables based on the arcs.
  - Helps identify values that need to be removed to maintain consistency.

- ‘clear_arcs‘ Method:
  - Clears the set of arcs, primarily used during the consistency-checking process.

- ‘remove_value‘ Method:
  - Removes a specific value from a variable’s domain and updates related data structures.

4 Data Structure Used

The project uses several data structures and algorithms to generate and solve Sudoku puzzles. Here's a brief overview of the data structures used and the reasons for their selection:

4.1 Lists

Lists in Python are used extensively throughout the project. They are used to represent the Sudoku puzzle itself, the values that can be filled in the puzzle, and the domains of the variables in the CSP (Constraint Satisfaction Problem) solver. Lists were chosen because they provide efficient access and modification of elements at any index, which is crucial for representing and manipulating the Sudoku grid.

4.2 Sets

Sets are used to represent the arcs in the CSP solver. An arc is a pair of cells in the Sudoku grid that are in the same row, column, or 3x3 box. Sets were chosen because they provide efficient operations for adding elements, removing elements, and checking if an element exists, all of which are used in the arc consistency algorithm.

4.3 Dictionaries

Dictionaries are used in the count_values function to count the number of occurrences of each value in the puzzle. Dictionaries were chosen because they provide efficient mapping from keys (the values in the puzzle) to values (the counts of each value).

4.4 Priority Queue (Heap)

A priority queue implemented as a binary heap is used in the CSP solver to select the next variable to assign a value to. The priority queue was chosen because it provides efficient operations to insert elements, remove the element with the highest priority, and update the priority of an element. This is used in the CSP solver to implement the Minimum Remaining Values (MRV) heuristic, which selects the variable with the fewest remaining values in its domain.

4.5 Tuples

Tuples are used to represent the return value of the is_valid function, which checks if a number can be placed in a specific cell of the Sudoku puzzle. Tuples were chosen because they are an efficient and convenient way to return multiple values from a function.

4.6 Booleans

Booleans are used in several places in the project, such as the domains of the variables in the CSP solver, the return values of functions like isv_valid and has_empty_cell, and the flags in the PQ class. Booleans were chosen because they are a simple and efficient way to represent binary (true/false) information.

5 GUI

This project is centered around a Python implementation of a Sudoku puzzle generator and solver, prominently featuring a user-friendly graphical user interface (GUI). The GUI seamlessly integrates various data structures and employs sophisticated algorithms to facilitate the generation and solution of Sudoku puzzles. Here are some GUI samples:

- User_initialization:

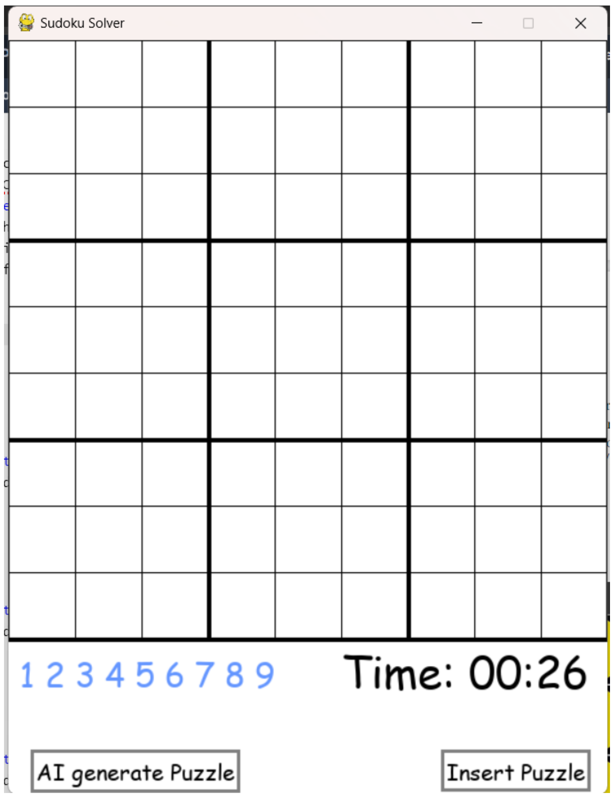
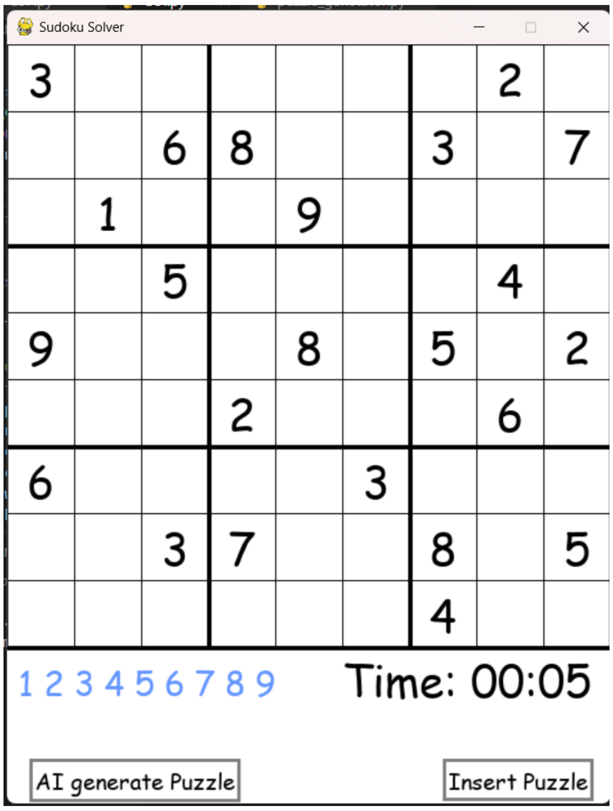


Figure 1: Empty Puzzle for initialization.

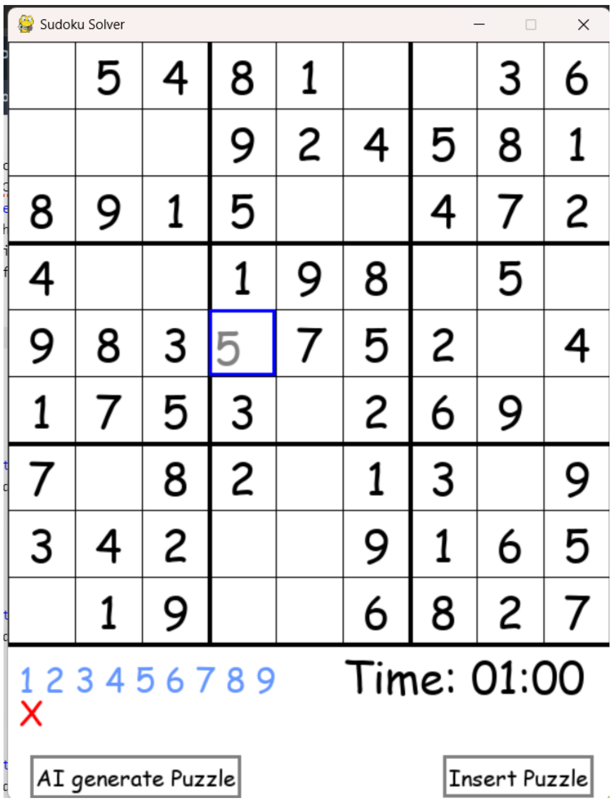
- Puzzle_generation:



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

Figure 2: Initialized Puzzle by the algorithm.

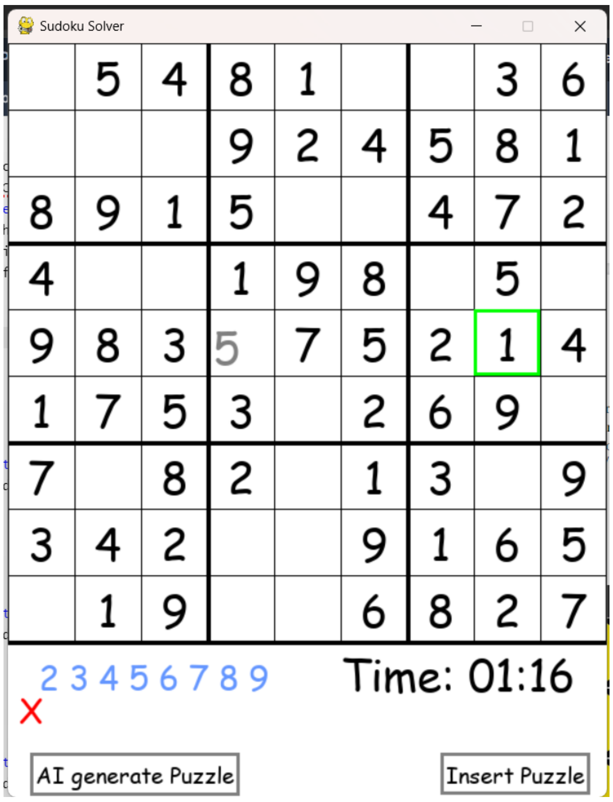
- Temp_insertion:



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

Figure 3: The temp insertion of a value that can be erased.

- Correct_insertion:

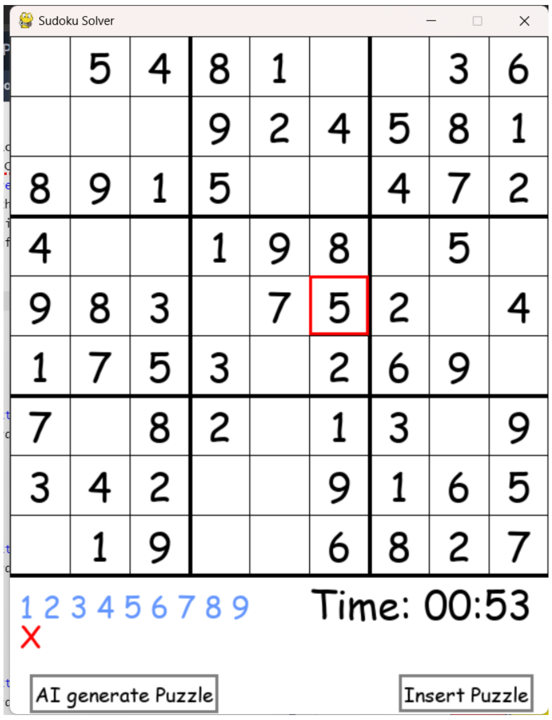


Time: 01:16

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

Figure 4: The correct insertion of a value.

- Wrong_insertion:

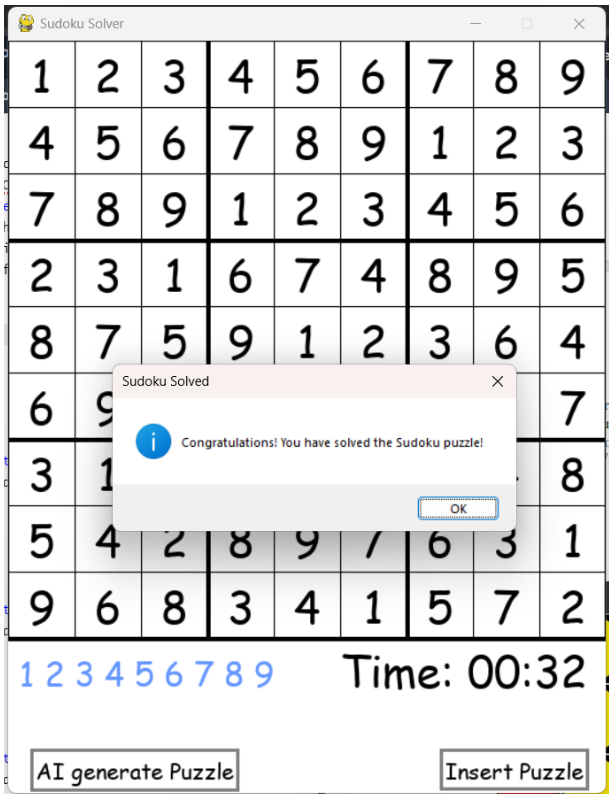


Time: 00:53

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

Figure 5: The wrong insertion and marking the conflicting node.

- AI_solve:



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

Congratulations! You have solved the Sudoku puzzle!

Figure 6: GUI when the game is finished.