Sudoku Solver

Intro To AI

Final Project

Report

Abdul Rehman Ansari - 24487

Burhanuddin Makra - 24460

Problem:

A sudoku puzzle is a grid of 9x9 cells, with each cell able to contain a number from 1-9. The goal of the puzzle is to fill the grid such that:

* Each row has numbers 1-9 with no repetition.
* Each column has numbers 1-9 with no repetition.
* Each 3x3 box has numbers 1-9 with no repetition.

Usually, some of the cells are already filled to ensure different solutions to each problem. We intend to develop an algorithm that given a sudoku puzzle, outputs the solution of that puzzle.

Goals:

* Develop an AI algorithm that solves a given sudoku puzzle of any size with as little time and exploration as needed.

Objectives:

* Develop the AI to solve 9x9 sudoku puzzles:
  + Develop a working heuristic function that minimizes the need to backtrack
  + Develop an algorithm that guesses only when required.

Methodology:

* The AI starts with a 9x9 sudoku puzzle.
* It adds numbers in empty cell blocks:
  + First, moves that can be confirmed to be correct are played.
  + Then a number is selected from a pool of possible numbers that do not contradict any numbers already on the board.
  + This is done by placing numbers 1 to 9 and checking whether that number is already in the row/column/box.
* If no number can be placed, we backtrack from the current state to a state from which we can get a new unique state.
* We do the above until we have a possible number to be placed in each cell and apply a heuristic function on each state.
  + The heuristic function returns the number of completed rows, columns, and boxes.
* The state that produces the highest number is picked to be the next state.
* This is repeated until the board is complete.
* Once the board is complete, we have solved the puzzle.

Note: This can be applied to any sized sudoku puzzle (4x4, 9x9, 27x27 etc)

Code explanation:

Sudoku.py (The game):

Constructor:

* Takes a NxN grid as input.
* Stores the grid.
* Calculates the size of the grid (4x4, 9x9, ….).
* Calculates the size of the little box.
* Initializes the lists that check whether a given row/col/box has a number or not.
* Initializes the list that keeps track of how many more of a number can be placed

Check Complete:

* Checks by going cell-by-cell whether the grid has been completely filled or not.
* Returns True if all cells are filled.
* Return False otherwise.

Find Box:

* Takes coordinates of a cell as input
* Outputs which box that cell is in.
  + The starting box is labeled 0, the second one on the row is labeled 2 and so on.

Make move:

* Takes coordinates of a cell and a number as input:
* Checks if that number can be placed in that cell or not
* If it can be placed, places it and updates the consistency lists.
* Else makes no move and alerts the player.

Undo move:

* Takes coordinates of a cell as input
* Clears the cell.

Print grid:

* Prints the grid in a way a sudoku puzzle is made

Heuristic:

* Counts how many rows are completely or almost filled.
* Counts how many columns are completely or almost filled.
* Counts how many boxes are completely or almost filled.
* Returns the sum of them.

Note: Almost filled means only 1 empty cell

Check Consistency:

* Goes cell-by-cell.
* Checks if a cell can contain a number while not violating any game rules.
* If a cell has no options, returns false.
* Else returns true.

Check Forced Num:

* Checks whether there exists a number that can only be placed once more.
* If there is a number, returns that number.
* Else returns -1 to signify that there is no such number.

Possible Move:

* Takes a number, and coordinates of a cell as input.
* Checks whether that number can be placed there or not. Does this by:
  + Checks if the cell is empty.
  + Checks if the row/col/box corresponding to that cell already has the input number or not.
* Returns true if the cell is empty and the number is not already in the row/col/box
* Else returns false.

Solver.py (The AI):

Constructor:

* Takes as input a sudoku object and stores it.

Single remaining:

* Checks whether there exists a number that can only be placed once more.
* If there is such a number, places it in the only place it can be placed
  + Because the other places are ruled out due to conflicts.
* Returns True if a number is placed.
* Else returns false.

Single option:

* Goes cell-by-cell and checks how many numbers can be placed in that cell
* If only one can be placed in it, places it.
* Returns True if a number is placed.
* Else returns false.

Single place:

* Goes row-by-row:
  + Checks in how many places each number can be placed in
  + If there is only one place, places it.
  + Repeats the above for columns and boxes.
* Returns True if a number is placed.
* Else returns false.

Make forced moves:

* Calls the above three functions.
* Returns True if a number is placed.
* Else returns false.

A Star Search:

* Initializes a queue with the starting state.
* Starts a while loop.
  + Each iteration is based on the first state in the queue.
* First, moves that can be confirmed to be correct are played.
  + By repeatedly calling the make forced moves function.
* Then all possible moves that keep the game consistent are generated and placed in the queue based on their heuristic score.
  + The higher the score, the higher the priority of them being considered in the next iteration of the while loop.
* The while loop runs until a completed state is found.

Limitations:

* The heuristic function has room for improvement.
  + We tried to include the number of moves, so more complete games are played first (steer it more towards Depth-first Searching). But, that just created a great amount of failing states if the guess was made right at the start of the game. Due to the sheer number of states being generated from a single state.
* There may be more ways to figure out moves that can be confirmed to be correct, but they are too analytical and there is no computational way to check all of them.
* If the starting grid is too empty (only a few cells are filled) and there is only 1 solution, then the AI has to guess right at the start. Due to the last 2 limitations.
  + This also creates a huge number of states that cannot yet be deemed to be failing
  + Note: The AI should reach the solution at some point of searching, but it tends to take a long time.