CSCI 3901: SOFTWARE DEVELOPMENT AND CONCEPTS

Assignment 4

Overview: Work with exploring state space

Strategy and Algorithm and Why it is efficient:

- Identified candidate solutions for each group separately beforehand. Consider a group, which has a condition 9* and is shaped as follows:

i	
j	k

  The possible values for {i,j,k} assuming a symbol set of {1,2,3} are as follows:
  [{1,3,3},{3,1,3},{3,3,1}].
  Of these solutions, we can immediately eliminate {1,3,3} and {3,3,1} because they violate row and column conditions; this step eliminates possible choices for this group. So I filtered the combinations.
- Instead of backtracking at the cell level, I choose to backtrack at the group level. Furthermore, I start filling in those cells which have the least uncertainty [by sorting the list of groups in ascending order of the number of solutions]
- While solving the puzzle, assume that *k* groups are already filled with a candidate solution and I'm trying out candidate solutions for the (k+1)-th group; in order to ensure efficiency, I automatically ignore those solutions of (k+1)-th group which are in conflict with the previously chosen solutions of k groups.

Classes:

- Group
  - Returns object for various groups(add, eq,diff,div)
- Mathdoku
  - Contains methods : loadpuzzle,solve ,readyToSolve , print. Here we load the puzzle along with groups and operations to be performed and then call solve method which solve the puzzle if its possible.
- Helper
  - Contains various methods : sum_combinations,mult_combinations, filter_solutions, isSolving, isSolved.Here we find the different combinations for multiplications and additions and also filter solutions. isSolving and isSolved are called to check for repetitive values in columns and rows.

Methods:

- Boolean loadPuzzle(BufferReader stream)

  - this method loads the puzzle as we store group objects and Group name (String) in a map string2group and then declare a puzzle of size n*n and also store operations to be done on group in a map called string2cells.

- Boolean solve:

  - in this method we solve the puzzle and return false if puzzle is not solvable

- Boolean readyToSolve:

  -checks various cases which should not allow puzzle to solve

- String print()

  - returns a string of the puzzle

- String choices()

  - returns the no of choices done during backtracking (at group level)

- Boolean tryout:

  -tries various combinations of solutions of groups to solve the puzzle

- sum_combinations:

  -returns various combinations of sol to Add group

Test Cases:

- Input Validations :
  Boolean loadPuzzle(BufferReader stream):
  - Stream is empty:return false
  - Stream is null: return false

- Boundary Cases:
  Boolean loadPuzzle(BufferReader stream)
  - Load a puzzle for one cell
  - Load a puzzle with big size
  - Loading a puzzle with no constraints
  - Loading a puzzle with more or less constraints than no of groups
  - Loading a n*n puzzle

Boolean solve()

- Solve a puzzle with one cell
- Solve a nxn puzzle

- Control flow

Boolean loadPuzzle:

- Loading a puzzle with standard size
- Loading a puzzle where no. of groups is equal to no. of constraints
- Loading puzzle when more spaces in constraints

Boolean solve:

- Solving a puzzle with no solution
- Solving puzzle with multiple soln
- Solve a puzzle with operators missing for some groups

Boolean readyToSolve:

- No of cells in div and sub(should ne 2)
- missing information for constraint of a group
- constraints are less than no of groups
- constraints are more than no of groups
- constraint have values equal to no of col or row of puzzle(n)

String print()

- Print a puzzle having no solution
- Print a solvedpuzzle
- Print an unsolved puzzle

Int choices ()

- Choices for a solvable puzzle(small in size or a large puzzle)
- Choices for an unsolvable puzzle

- Data flow:
  - Calling load puzzle multiple times with different streams
  - Calling solve method multiple times
  - Calling readyToSolve method before solve
  - Calling solve method before loadpuzzle

- Calling print method before loadPuzzle method
- Calling print before solve method
- Calling print method after calling loadpuzzle and solve method
- Calling choices method before solve method and after solve method
- Calling solve method after load puzzle
- Calling choices method after calling solve multiple times
- Calling choices method when puzzle loaded again
- Calling loadpuzzle then solve then print method and then again load and print
- Calling readyToSolve after solve method and before solve method

Assumptions:

- If a constraint for a letter is there which letter is not in puzzle , solve method returns false
- Empty lines and line with whitespace characters are skipped
- There can be only 2 cells for add and subtraction
- if puzzle is not solvable return 0 choices
- if all constraints are not given readyToSolve and solve method return false
- Strings are case sensitive
- If any group is not solvable print returns the original loaded puzzle
- Return false if any letter is in puzzle but not in constraints