FINAL REPORT

THE UNIVERSITY OF HONG KONG
COMP3258: FUNCTIONAL PROGRAMMING
FINAL PROJECT (JIGSAW SUDOKU GAME)

NAME: HU QIYUN   UID: 3035449396

1. How to build the project:

I develop my project on macOS, please use equivalent steps if you are using other OS.

a. Unzip the zip file, you then can get a folder.

b. “cd ../YourPath/FinalProject-sudoku/src”
   Open up a terminal and change into the project directory.

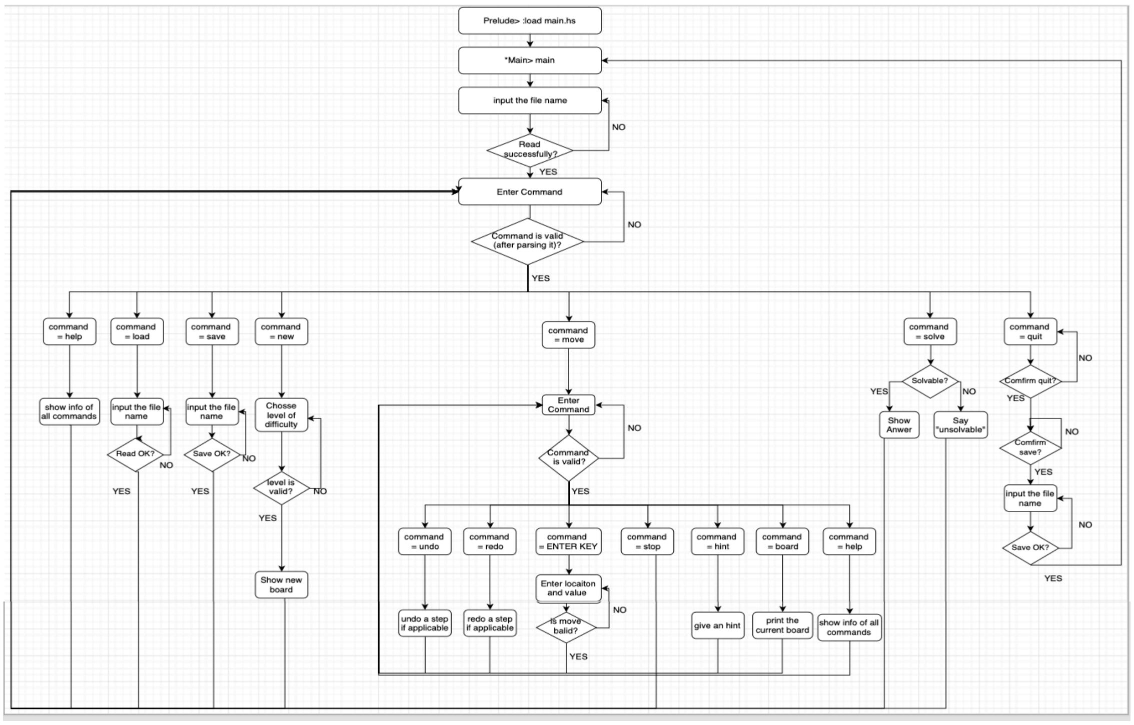
c. “make main”
   To compile the file in terminal.
   What happens in Makefile is “ghc --make main.hs -o main”

d. “make run”
   To run the game in the terminal.
   What happens in Makefile is “./main”

2. The functionality of the program

My sudoku game has various functionalities, If being fully explained in text it would be verbose so I will make some essential explanations and give a flow-chart of my game to make it clearer.

This is the flow chart: if you think it is not clear enough, you could refer to the “sudoko-flowchart.pdf” under the same folder.



a. **Essential explanation 1:**

My game has an Outer Loop for the whole game, and a nested Inner Loop for continuously fill up the cells.

Outer Loop commands you can choose:

"move" - To fill up a point, once in the loop of move, you would be offered more functionalities, such as redo, undo, hint, etc. → to enter the Inner Loop

"save" - To save the current board of the sudoko game to an indicated file.

"load" - To load a board of sudoko game from an indicated file.

"quit" - To quit the current sudoko game after your confirmation. You would also be offered a chance to save the current board of the sudoko game to an indicated file.

"new" - Generate a new random sudoku board, you can chooes the level of difficulty.

"solve" - Show a solution of the current sudoku game, without ending up the game, you could still work on your game.

Inner Loop for filling up cells commands you can choose:

"stop" - To stop the move, jump out of the loop of filling.

"hint" - It prompt you with the of possible values could be filled into the next blank cell.

"board" - To print the board to the console.

"undo" - To undo the fill-cell action if there exists more than zero previous actions.

"redo" - To redo the fill-cell action if there exists more than zero undo-ed actions.

"help" - Display the commands and their usage

b. **Essential explanation 2:**
The file to be saved/loaded are all stored in the "maps" folder, in the game, you only need to specify the name of the file (instead of path) to save/load. DO NOT delete "template_sol.txt" in maps folder since it is for generating random new board.

c. **Essential explanation 3:**
Make sure you enlarge the width and height of the console for the proper and nicer display or game board and command information display

3. Data structures for representing Jigsaw Sudoku boards.

a. **To represent a Cell in a sudoku game, I make use of the "data":**

   data Cell = Cell Int Int Int Int deriving (Eq)
      Four parameters in the data constructor of Cell represents
      column index (Int, 0-8 inclusively),
      row index (Int, 0-8 inclusively),
      block number (Int, 0-8 inclusively),
      value (Int, 0-9 inclusively, 0 means the cell is empty, otherwise is filled up with the corresponding value), respectively.

b. **To represent a Jigsaw Sudoko boards, I make use of the "type" and Cell in 2.a:**

      Type SudokuBoard = [[Cell]]

4. Error cases and ending.

a. The code handles all exception cases that may happen during read/write file actions. That is, if a file does not exist during a read action, or is locked during a write action, etc., the system will prompt you with the information of the exception and/or error cases, then it asks the user to redo the action until the action successes.
   This exception handling avoids the game from quitting unexpectedly.

b. The code handles all cases when user enter an invalid command.
   If the command is totally wrong, then it asks the user to re-enter their command until it is valid.
   If the command is partially wrong, meaning a command may be a sub-string of user input, then it prompts the user with a similar valid command and then asks the user to re-enter their command until it is valid.

c. For Ending, unless user quits from ghci or by entering a "quit" command (without quitting from ghci), the game would not quit. When a sudoku game is completed, the system prompts the user with "Congratulations…" and allows the user to "load" a new file or make a "new" random board to start a new game by the user's will.

5. Additional features and how their implementation works.

a. **Additional Feature 1 - Undo:**

   It allows the user to undo a previous fill-cell action if there is ≥ 1 records of fill-cell action, by entering "undo" command during the loop of filling cells.

For the action the user made, it is recorded to a list of undo actions, when system executes the “undo” command, we retrieve and then remove value from this undo list. The undo-ed action is then recorded by the redo list.

Undo list and redo list are both FILO.

b. Additional Feature 2 - Redo:

It allows the user to redo a previous undo-ed action if there is ≥ 1 records of undo-ed action, by entering “redo” command during the loop of filling cells.

For the undo action the user made, it is recorded to a list of redo actions, when system executes the “redo” command, we retrieve and then remove value from this redo list. The redo action is then recorded by the undo list.

Undo list and redo list are both FILO.

c. Additional Feature 3 - Solver:

The system provides a solver for the user.

The solver prints a sudoku board to the console if there exists a solution, otherwise tells the user the current board is unsolvable, by entering “solve” command during the loop of game body (outside the inner loop of filling cell).

It is noticeable that I intentionally just print the solution board without actually helping the user finish the current game, thus, the user still has a chance to finish the game.

The algorithm behind is DFS, but for each empty cell, it only tests for possible values instead of 1-9. The solving time could be fast for sudoku boards having around <50 empty cells, and could be one/two minutes for sudoku boards having around 53-57 empty cells, and it is not recommend to solve sudoku boards having > 60 empty cells.

d. Additional Feature 4 - Hint:

The system will give a hint to user, showing the information of all possible values that can be filled into the next empty cell (typewriting direction), by entering “hint” command during the loop of filling cells.

e. Additional Feature 5 - Generate New Board:

The system will generate random board to user, by entering “new” command during the loop of game body (outside the inner loop of filling cell).

It is noticeable that user can select a difficulty level (easy/medium/hard), easy = 30 empty cells, medium = 40 empty cells, hard = 50 empty cells.

The problem I met:

(1) If generating a random board from a totally empty board:
Pros: the block shape could be random,
Cons: it takes a long time to test if a generated board is solvable, but actually it has a low possibility that a random board is solvable, and each testing could be slow for a board with lots of empty cells.

(2) If generating a random board from a completed board:
Pros: it is sure to be solvable.
Cons: it is not possible to generate a random completed board.

My choice:

I generate the board from a completed board (a known stored file), with number being switched randomly and removed randomly. But the drawback is block shape is not random.

f. Additional Feature 6 - Good-looking user interface:

Although using terminal, the board is marked with column index and row index, make it convenient for user to identify a cell.

```
         Column Index
             0    1    2    3    4    5    6    7    8
Row Index  _____________________________________________
          |                                             |
       0  |  .    8 |  .    .    .    3 |  .    .    .  |
          |          ----------                         |
       1  |  1    .    .    . |  7    . |  .    .    .  |
          |          ----------          -----          |
       2  |  7    . |  . |  . |  .    1    . |  .    .  |
          |     -----          --------------------     |
       3  |  5 |  .    . |  .    . |  .    .    7 |  .  |
          |-----                    -----          -----|
       4  |  .    .    3 |  .    .    . |  .    2    .  |
          |-----          -----                    -----|
       5  |  . |  .    8    . |  .    . |  6    . |  .  |
          |     --------------------          -----     |
       6  |  2    . |  .    .    . |  . |  . |  .    3  |
          |          -----          ----------          |
       7  |  8    .    7 |  .    . |  .    .    .    .  |
          |                         ----------          |
       8  |  .    .    . |  8    .    .    . |  .    7  |
          |_____________________________________________|
```