University of Hong Kong

COMP3258 Final Project

Jigsaw Sudoku Game GUI

Developed in Haskell

By CHEN Xing Quan Hansen (3035440912)
Department of Computer Science

For Course COMP3258 Functional programming

December 31, 2019

Table of Contents

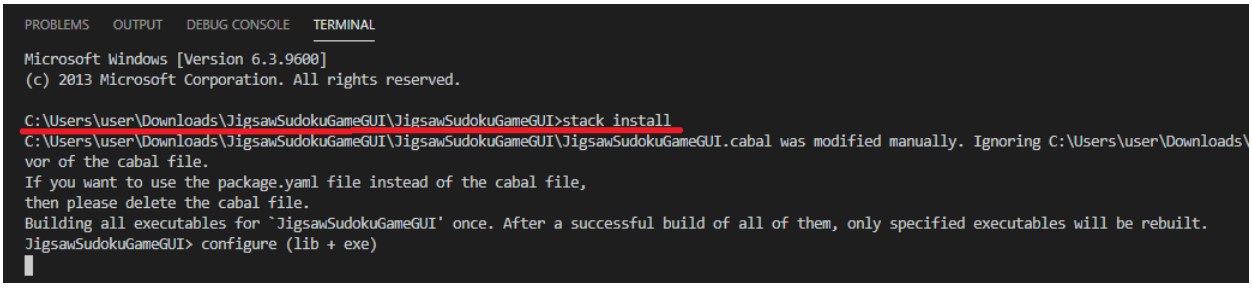
1. Build the Project

OpenGL (https://www.opengl.org/), GLUT (http://freeglut.sourceforge.net/) and Haskell (https://www.haskell.org/platform/) must be installed for building and running this project

Normally Windows and Mac do not need installations for OpenGL and GLUT.

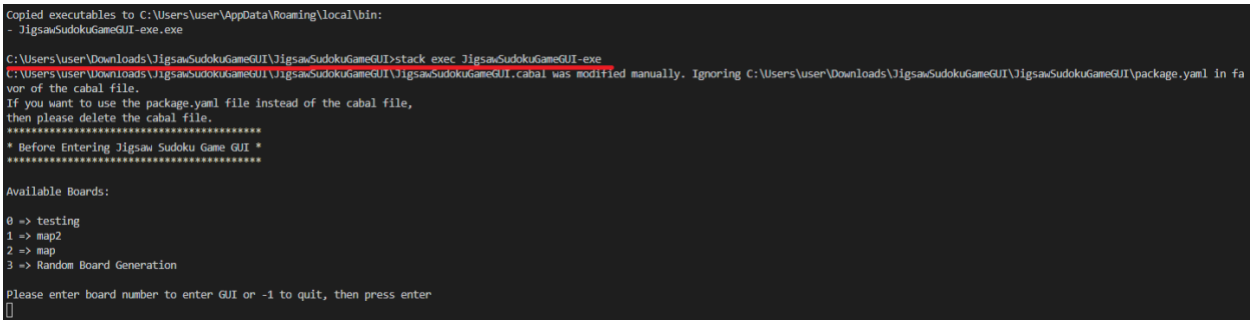
0. On Windows or Mac, get the project file by unzipping the zip file or by running "git clone https://github.com/Hansen-chen/JigsawSudokuGameGUI"

1. Under the project root folder "JigsawSudokuGameGUI" run command "stack install"



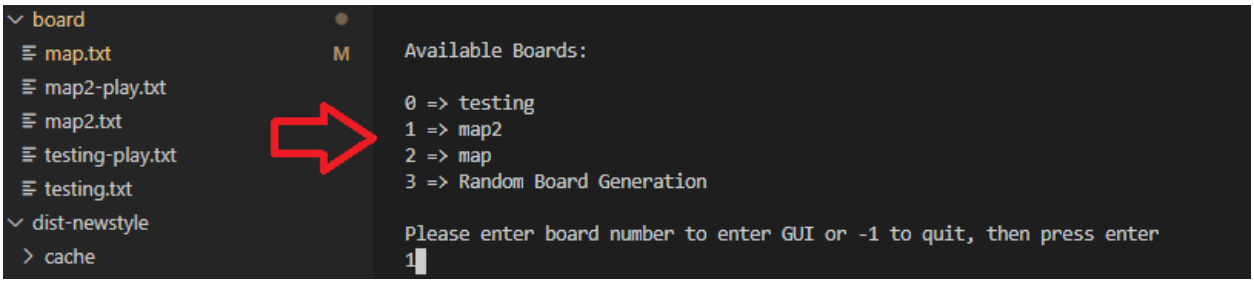
If you encounter error in step one, run command "stack clean"

2. After that, under the project root folder "JigsawSudokuGameGUI" run command "stack exec JigsawSudokuGameGUI-exe" to start the game



2. Functionalities and Corresponding Game play

i) Load File



This game stores **all board files** (like map.txt provided in moodle) and **corresponding game progress file** (board file name with suffix "-play" like "map-play.txt") under **"board" folder** of this project root folder. If You modified board file, you must delete corresponding game progress file (board file name with suffix "-play" like "map-play.txt") to start new game. Otherwise error will occur.

You must only put in board file(txt file) which has **name containing only at most 10 alphabet or digit characters** in to the "board" folder of this project root folder.

All boards and random board generation option will be listed out.

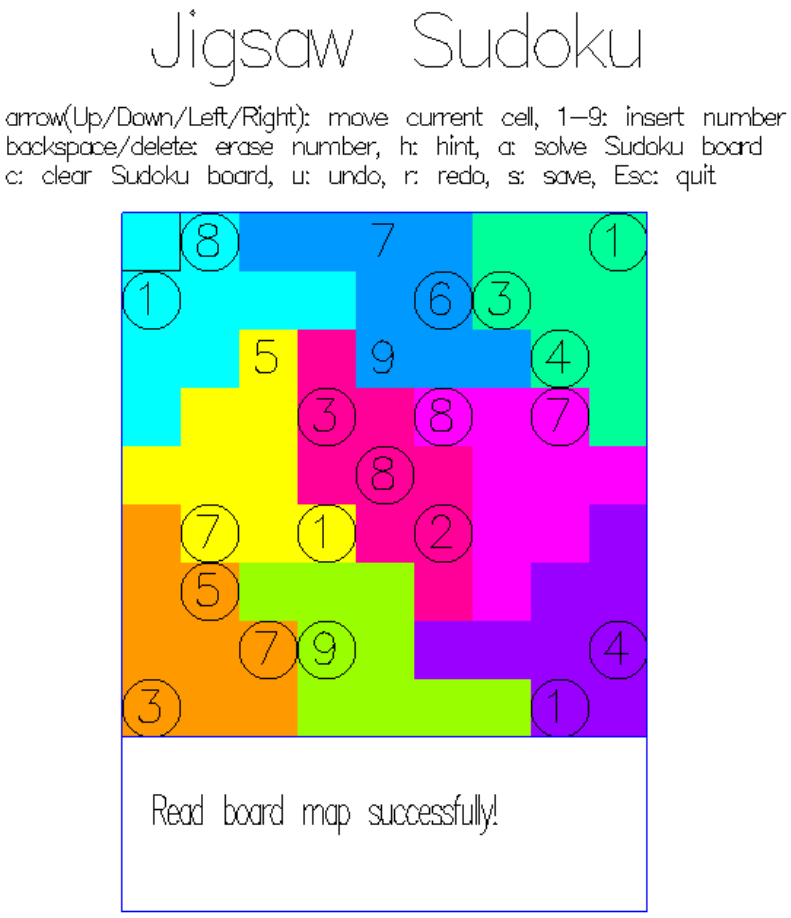
```
****************************************
* Before Entering Jigsaw Sudoku Game GUI *
****************************************

Available Boards:

0 => testing
1 => map2
2 => map
3 => Random Board Generation

Please enter board number to enter GUI or -1 to quit, then press enter
2
Loading Jigsaw Sudoku Game board ...
Entering Jigsaw Sudoku Game GUI...
```

After entering the board number shown under "Available Boards", e.g. 2 for board in "map.txt", you will enter full screen graphic user interface(GUI) with message box saying "Read Board boardname successfully!" and short cut key reminder and load the corresponding game progress as shown in next page. If no option number is entered, the message will be displayed again. If -1 is entered, the program will stop.



ii) (Extra Features) Graphic User Interface (GUI)

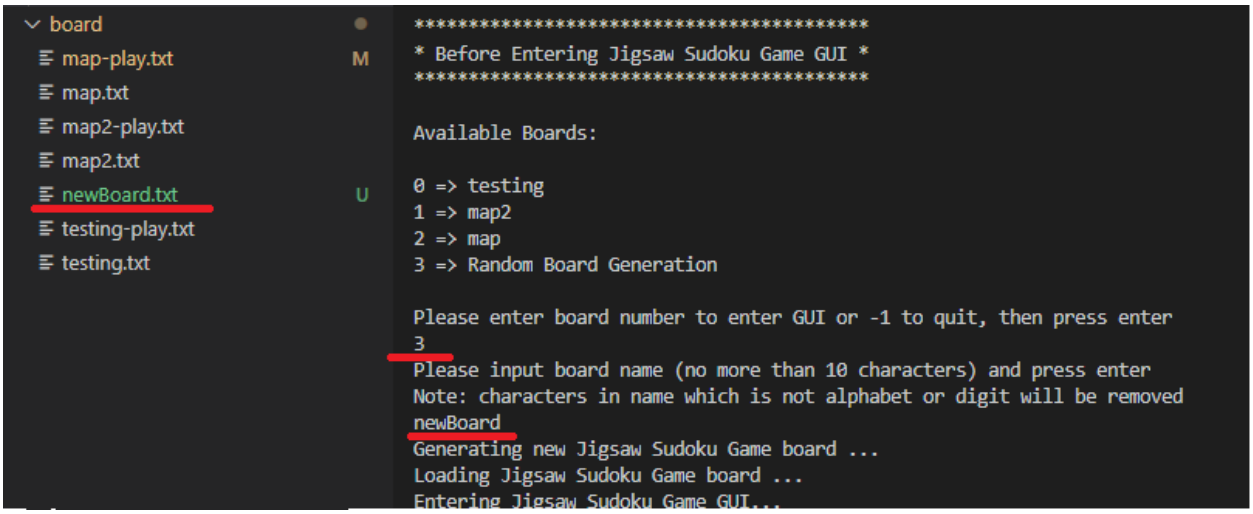
As shown in previous functionality and next all functionalities, a good looking and informative full screen GUI will be in the game play.

This GUI contains short cut key reminder below the game title "Jigsaw Sudoku".

Below short cut key reminder is the main board, **the original number in the board is circled while the inserted number by player is not. The cell surrounded by black square is current cell player is in**. Different color indicates different block. The colors for corresponding blocks are randomly generated.

The message box is at the bottom, it will show corresponding message after player press short cut key, load board or generate board.

iii) (Extra Features) Solvable Random Jigsaw Sudoku Board Generation



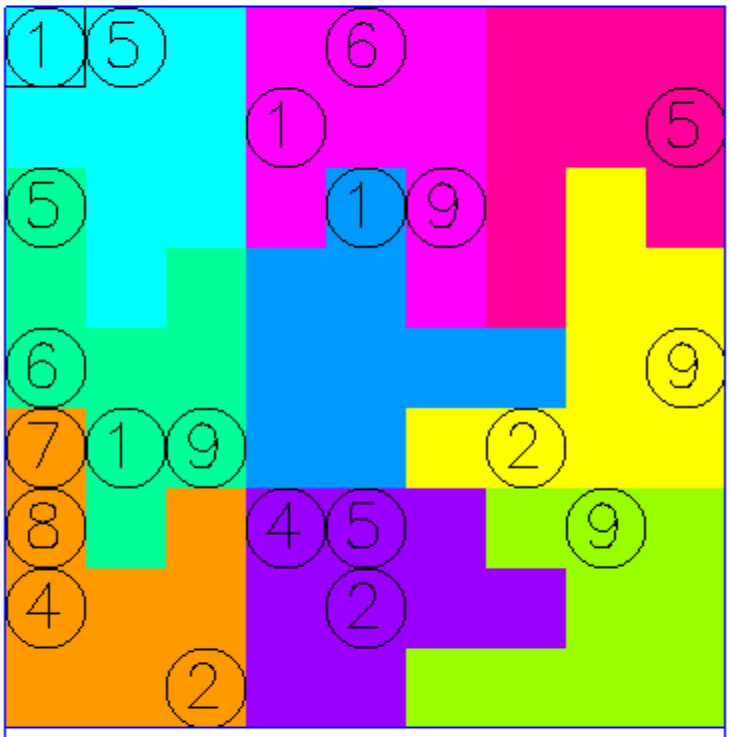
After you enter random board generation option number shown below "Available Boards", you will need to enter **board name containing only at most 10 alphabet or digit characters**, any character violates this rule will be removed, empty name will be named as "unname".

A board file(.txt) containing the board name will be created under board folder as shown(newBoard.txt). If you save game progress of this board, a game progress file mentioned previously will be created too(newBoard-play.txt).

After waiting for few seconds for random board generation, you will enter full screen graphic user interface(GUI) with message box saying "Generate Board boardname successfully!" and short cut key reminder and load the corresponding game progress as shown in next page.

Jigsaw Sudoku

arrow(Up/Down/Left/Right): move current cell, 1–9: insert number
backspace/delete: erase number, h: hint, a: solve Sudoku board
c: clear Sudoku board, u: undo, r: redo, s: save, Esc: quit

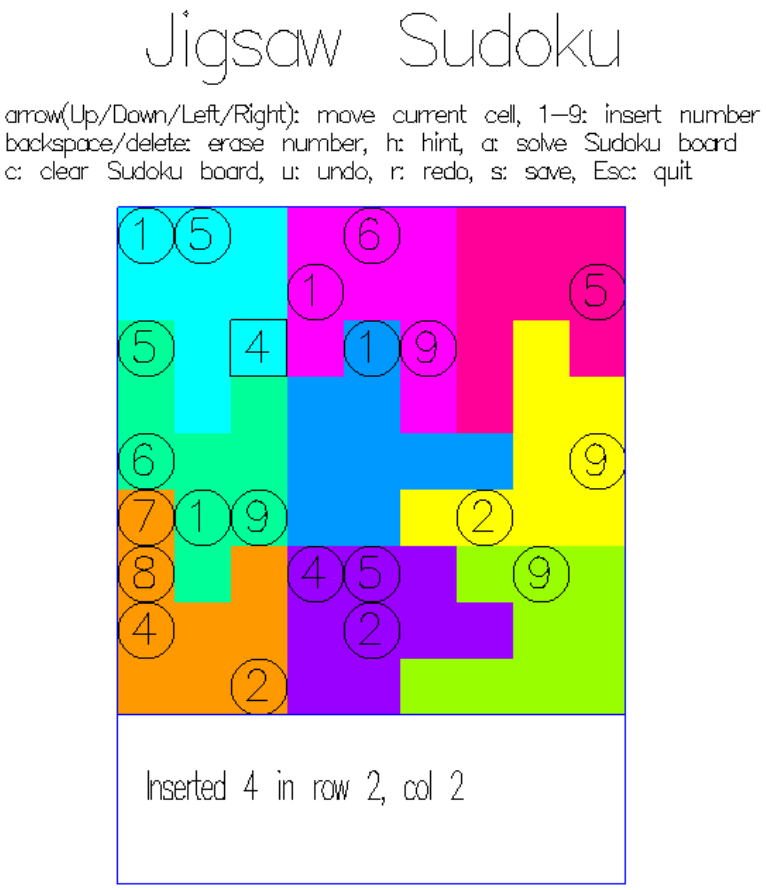


Generate board newBoard successfully!

iv) Make Move

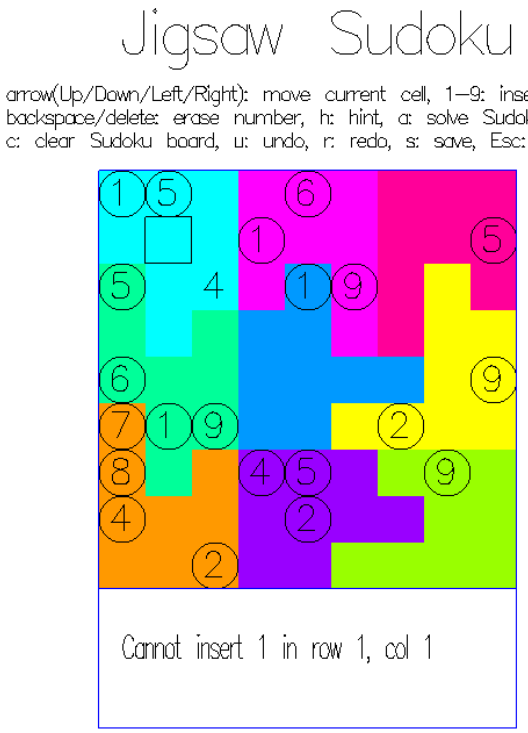
After entering GUI, you can press arrow (Up/Down/Left/Right) in the keyboard i.e., (↑ / ↓ / ←/→) to move the current cell you are in in the board which is represented by a square with black border.

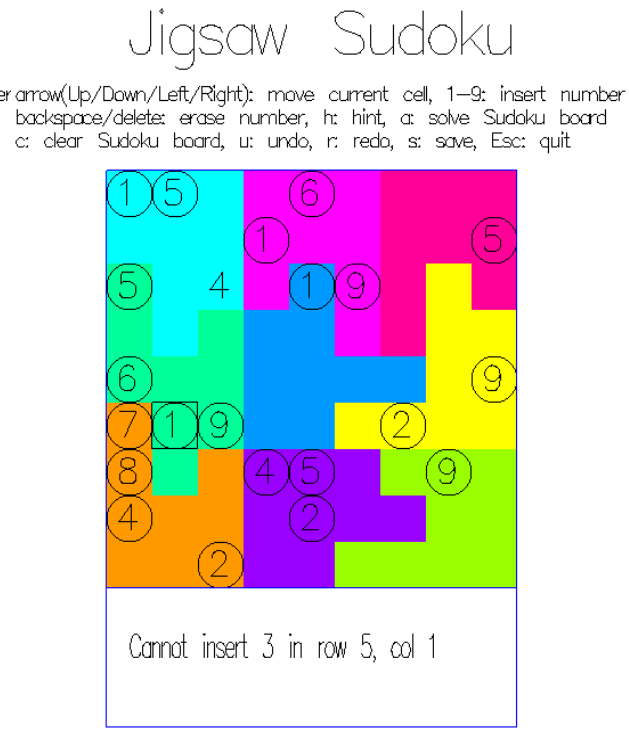
Then you can press 1-9 in keyboard to insert number in current cell. Message box will say "Inserted number in row y col x as shown below (Inserted 4 in light blue block in upper left hand corner).



v) Error Handling

When you make a move, the game will detect errors including violations of the Sudoku constraints or make move in original number or inserted number in the board. The error message will be shown in the message box as shown below. Also, as the current cell you are in can only be inside the board, checking for ranges of numbers and positions is automatically done.

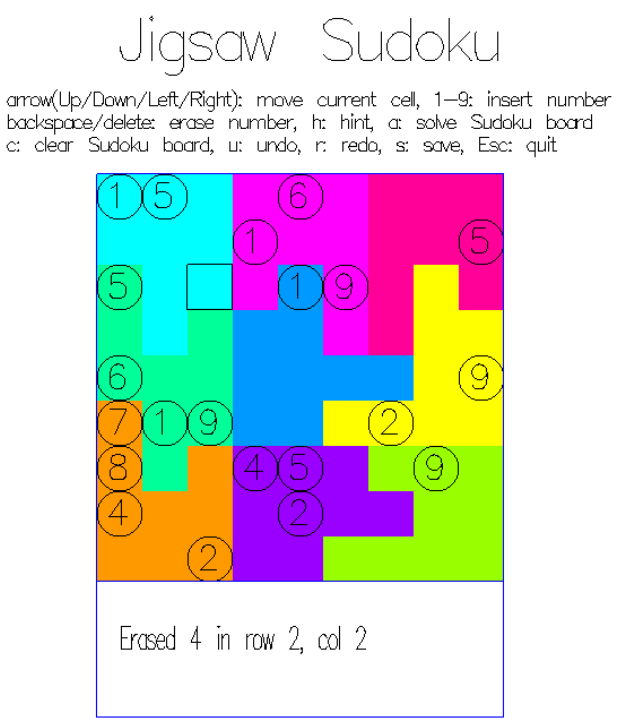


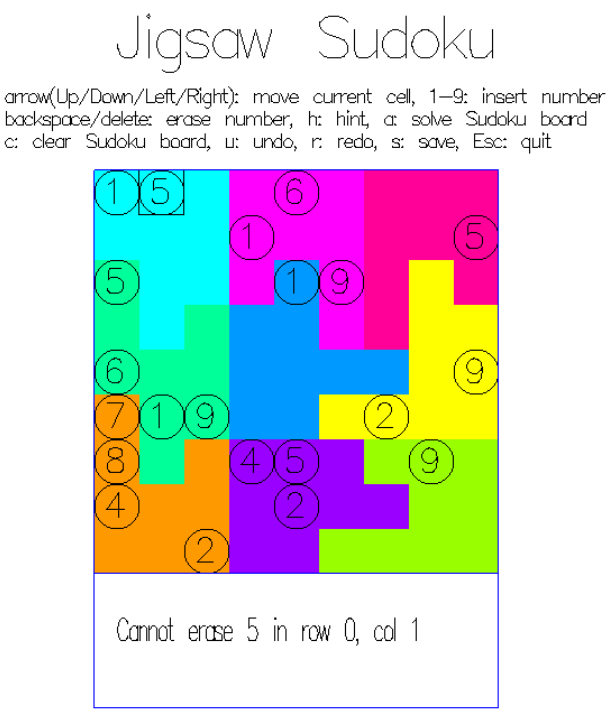


vi) (Extra Features) Erase Number with Error Handling

You can press arrow (Up/Down/Left/Right) in the keyboard i.e., (↑ / ↓ / ←/→) to move the current cell you are in in the board. You can erase the inserted number in current cell in the board by pressing **backspace or delete** in keyboard. The corresponding message will be shown in the message box as shown below. (Erased 4 inserted in previous functionality)

When you erase a number, the game will detect errors including erasing original number in the board. The error message will be shown in the message box as shown below.

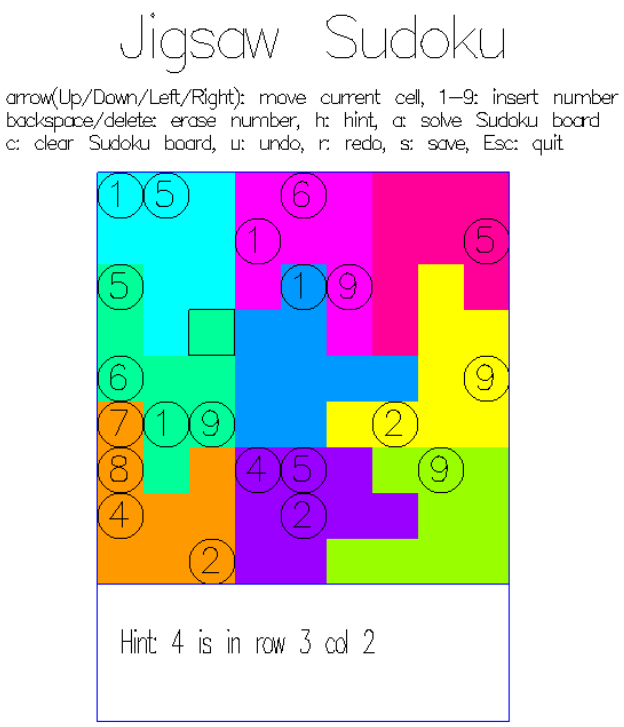




vii) (Extra Features) Hints

You can press arrow (Up/Down/Left/Right) in the keyboard i.e., (↑ / ↓ / ←/→) to move the current cell you are in in the board. Then you can press **h** in keyboard to get hint of current cell. If the current board is solvable, the solution of current cell will be solved and displayed in the message box as shown.

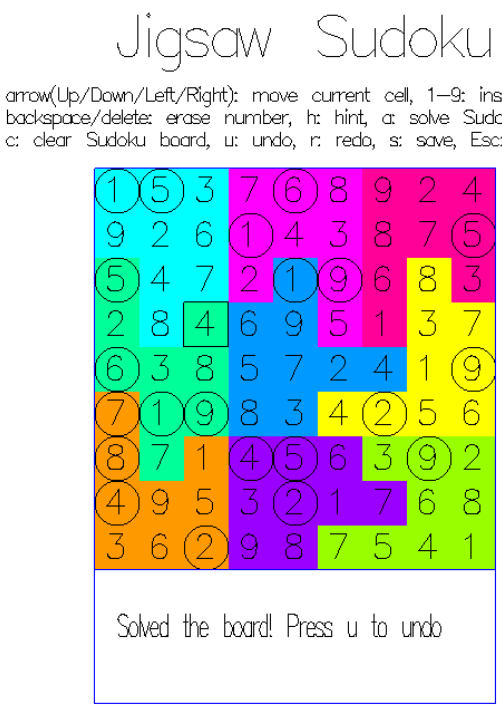
Otherwise, "Hint: This board has no solution" will be displayed in the message box as shown.

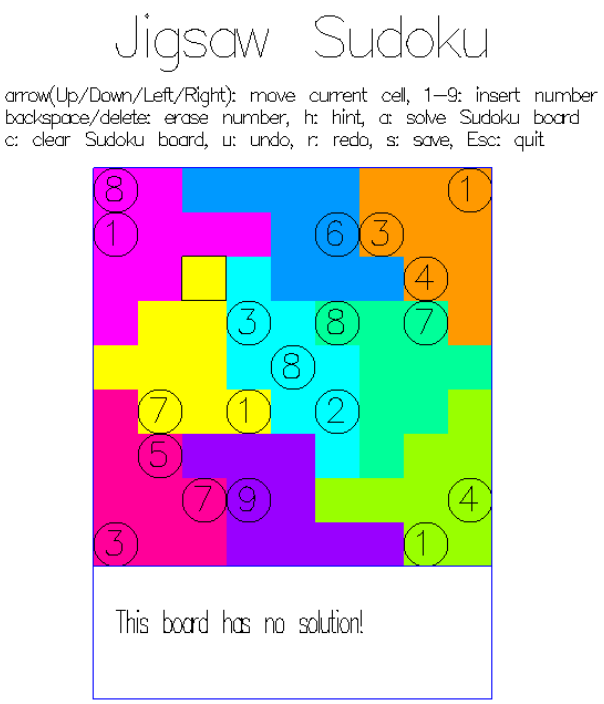


viii) (Extra Features) Solver

If the current board is solvable, you can press **a** in keyboard to invoke solver and get the answers of current board. Corresponding message will be displayed in message box as shown.

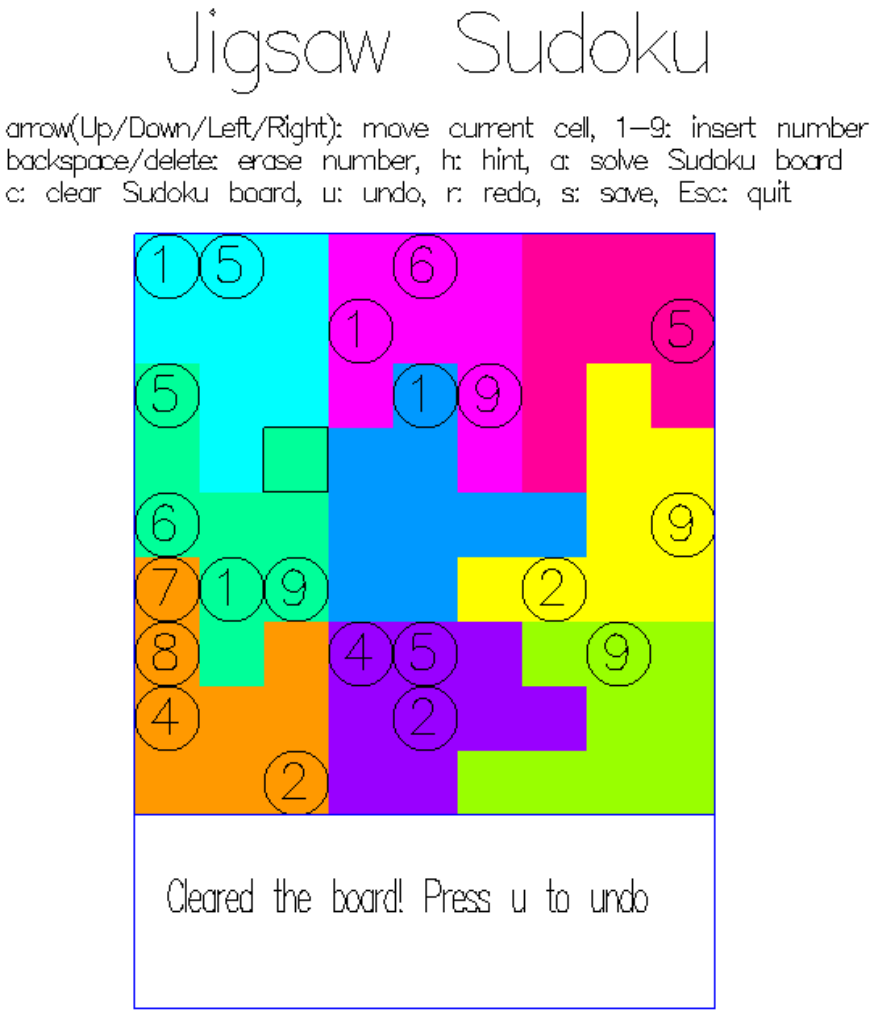
Otherwise, "This board has no solution!" will be displayed in message box as shown.





viii) (Extra Features) Clear all

You can press **c** in keyboard to clear all inserted number and restore the original board. Corresponding message will be displayed in message box as shown.



viii) (Extra Features) Undo / Redo

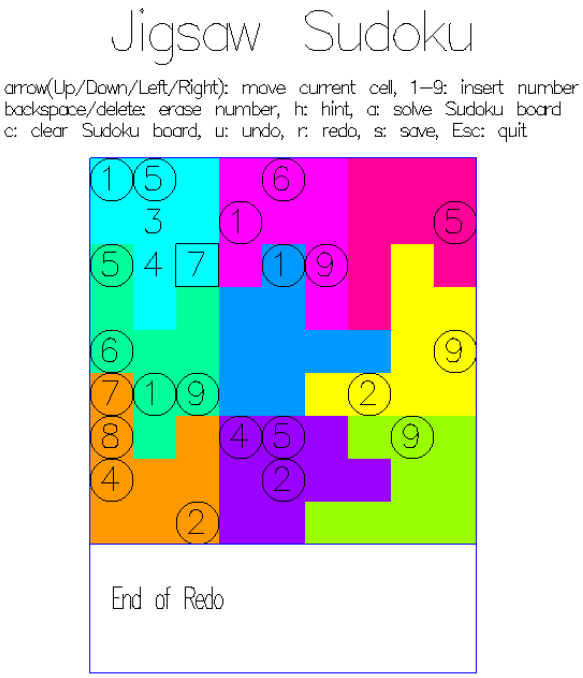
You can press **u or r** in keyboard to undo or redo the action you performed in the game.

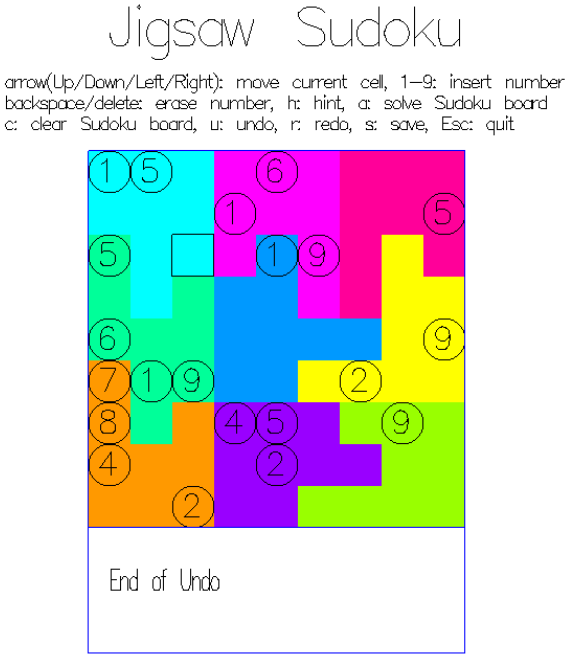
The actions supported by undo/redo operations are "make move" (insert number, including insertions trigger error message), "erase number", "solver" and "clear all".

Note: If you perform a new action (e.g. insert new number) when you are performing undo/redo and there are still actions you can perform when you press u or r, the remaining actions you can redo will be deleted as convention.

Corresponding message will be displayed in message box when you press u or r.

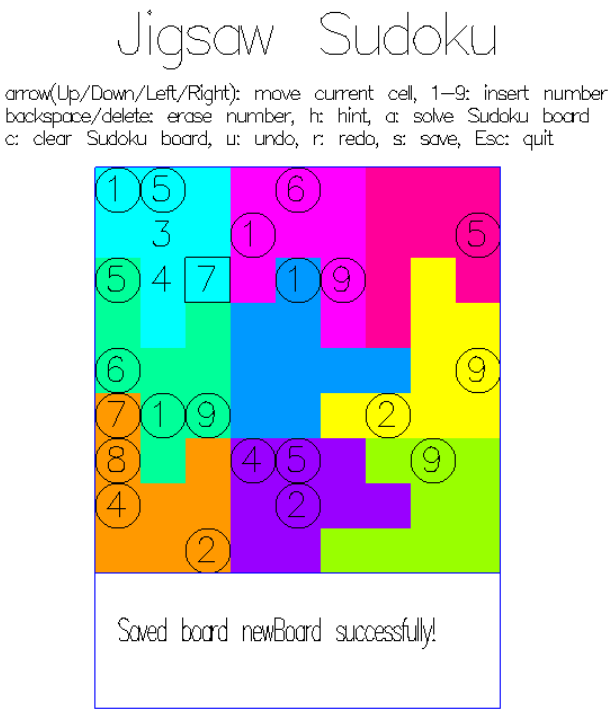
When there is no action you can redo/undo, "End of Redo" or "End of Undo" will be displayed in message box as shown.

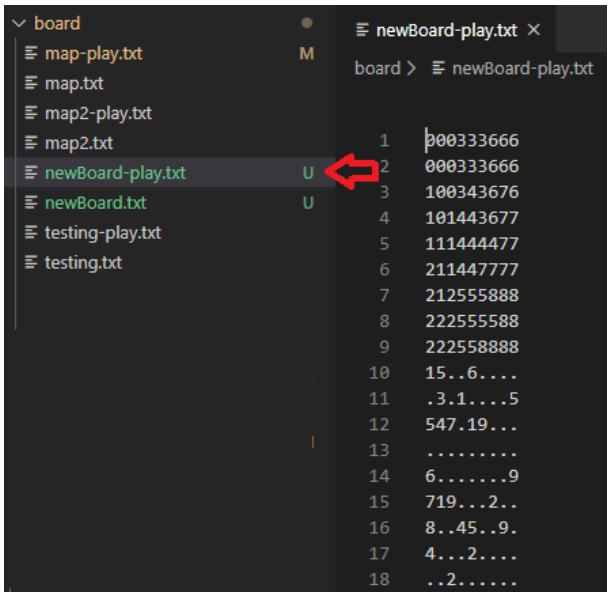




viii) Save

You can press **s** in the keyboard to save current game progress. A txt file named in board name with suffix "-play" will be generated under board folder under root folder of this project. Corresponding message will be displayed in message box as shown.





ix) Quit

You can press **Esc** in the keyboard to quit the game and GUI.

3. Choice of Data Structures

My choice of data structure for Jigsaw Sudoko boards is two Int Data.Array.

The reason why I choose Int Data.Array is that all the data need to be stored is an integer and most of the actions in the game need to access data by row and column.

Thus I use one 2d Array to represent blocks and one 2d array to represent numbers in the board.

As I uses gloss hackage to implement GUI, I self-define a Game data type containing the current game progress of current Jigsaw Sudoko board, a string message for GUI to display current message, a list of Gloss.Data.Color for representing different blocks, original board (including original number and block) to display in GUI and finally a string filename for save action.

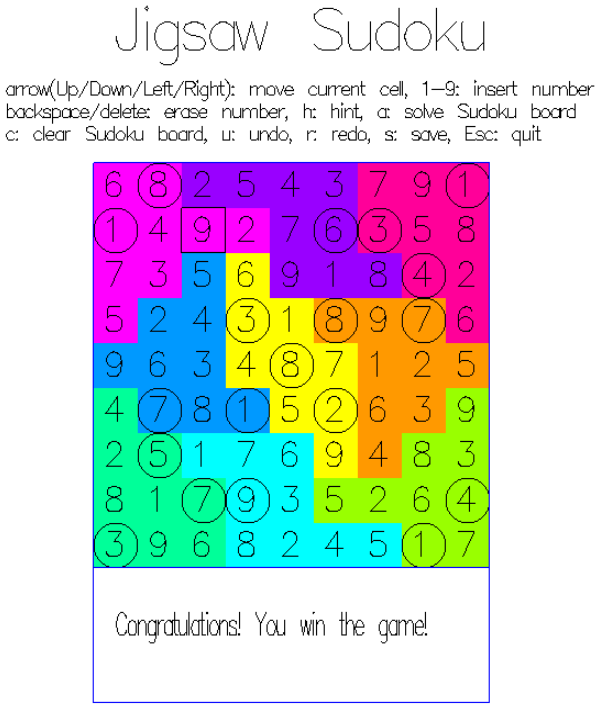
Furthermore, I self-define a GameState data type to contain the current game, a tuple (Int, Int) to store current cell, a board object containing current board's solution (a 9*9 2d array full of (-1) if the current board do not have a solution), current board's initial number (including original number and numbers player previously inserted, a list of action in ((Int,Int),Int) format indicating number insertion, solve((-99,-99),-99) or clear all((99,99),99) and finally a Int game pointer for implementing redo/undo to locate and reconstruct current game board in moves array.

4. Deal with Error Cases and Ending

For error handling, the gloss GUI filters out violations of ranges of numbers and positions as players can only move current cell within current board by setting number constraint (0-8) for moveCurrentCell function when player presses arrow (Up/Down/Left/Right) in the keyboard i.e., (↑ / ↓ /←/→) for moving currentCell.

For violations of the Sudoku constraints, check function in JigsawSudokuControl.hs check whether the number exists in the same row or column by extracting the numbers using map function and assocs function of Data.Array. Then jigsawBlockCheck function in same .hs file checks whether the number exists in the same block by extracting extracting the numbers using map function and assocs function of Data.Array. Error handling examples can be found in section 2 Functionalities and Corresponding Game play.

For game ending, the game will check whether the current board is the same as the solution solved by game solver every time player insert number. If the player successfully solves the board by inserting number by himself/herself, congratulation message will be displayed as shown.



However, if user finish the board by invoking the solver to solve the whole board, congratulation message will not be displayed as shown in section 2 Functionalities and Corresponding Game play.

5. Additional Features Implementation

This game has the following extra features which are also covered in section 2 Functionalities and Corresponding Game play.

- Graphic User Interface (GUI)
- Erase Number with Error Handling
- Clear all
- Undo / Redo
- Solver
- Hints
- Solvable Random Jigsaw Sudoku Board Generation

Examples and screenshots of the above extra features can be found in section 2 Functionalities and Corresponding Game play. These extra features implementation will be covered loosely one by one as follow.

Graphic User Interface (GUI)

I use haskage Gloss which is the most popular GUI hackage in terms of numbers of download. I render the GUI by three functions renderUI which construct the layout of GUI, inputHandler which takes keyboard input from player and lastly updateGame which is not used.

Erase Number with Error Handling

I use the same function "make move" functionality uses and add a case (-1) because in my self-defined jigsaw Sudoku board means empty. Also I add checking on not allowing player to erase original number of jigsaw Sudoku board.

Clear all

I also use the same function "make move" functionality uses and add a case change cell in column 99, row 99 to 99 because this number will never be reached and it is safe for this features to not mixed up with "make move" functionality. Then I change the board recording the game progress back to original board to remove all inserted number

Undo / Redo

Every time player insert number, invoke solver to solve board or clear all inserted number, I add that action to "moves" list and I add a Int type variable gamePointer to record current game state (current position of the game in "moves" list). When the player performs undo/redo, function jigsawSudokuGameUndo or jigsawSudokuGameRedo will decrement or increment gamePointer by one and call jigsawSudokuGameReconstruct to perform actions to original board by list of "moves" until reaching gamePointer position.

Also, by convention, if gamePointer does not reach the end of list "moves" but new actions are performed, the remaining of list "moves" are dropped.

Solver

As solving Sudoku game can be constructed as finding solution to a set cover problem, I use hackage set-cover and adopt codes from example of using hackage set-cover in the following link to find solution for current board, if no solution is found, return 9*9 Int Data.Array filled with (-1) as (-1) indicates empty my self-defined data structure for Jigsaw Sudoku Board

https://hub.darcs.net/thielema/set-cover/browse/example/Sudoku.hs

Hints

I first invoke solver by calling function solveGame to find solution for the current board and then save the result to solution field of self-defined data structure GameState when loading the game. Then when player press h for hint of current cell, I will generate hint message by accessing solution Int Data.Array using current cell's row and column number. If the number return is (-1), then current board do not have solution, message box will display this message. Otherwise, correct answer of current cell will be displayed in the message box.

Solvable Random Jigsaw Sudoku Board Generation

As constructing Sudoku board can be constructed as constructing a set cover problem, I use hackage set-cover and adopt codes from example of using hackage set-cover in the same link used in previous section "solver". Keep randomly generate Jigsaw Sudoku Board until board with connected blocks and only one solution found, otherwise will call function itself again to

generate again. Also, I set time out for generating because solve Sudoku function will not stop until it find at least one solution.