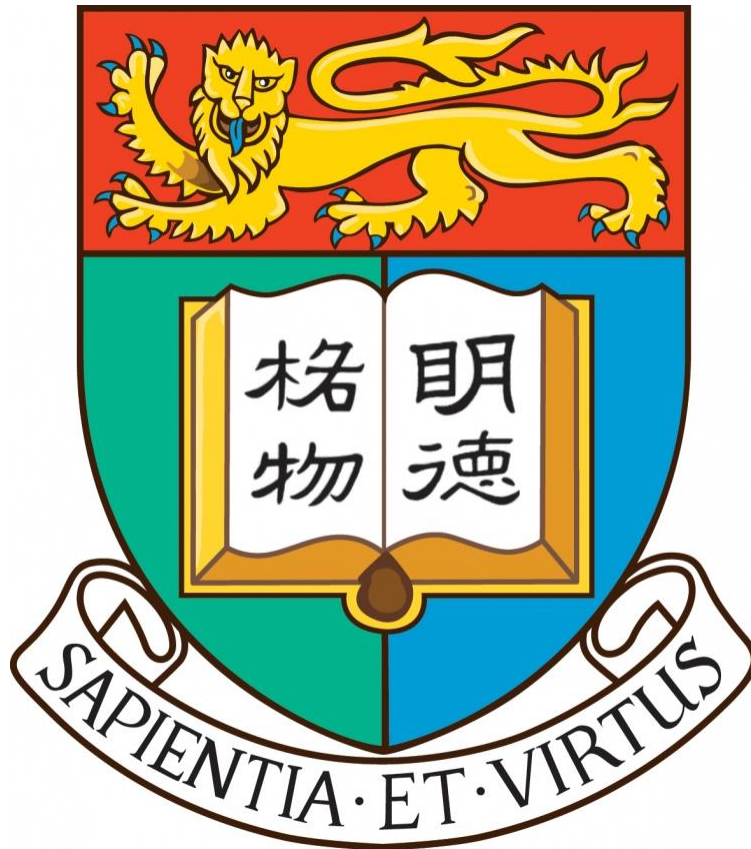


COMP 3258 : Functional Programming

Final Report: Jigsaw Sudoku



UID: 3035341922

Subhayan Roy

Contents

Building the game.....	3
Game Interface	3
Basic Functionalities.....	4
Load a file.....	4
Save a file.....	4
Make a move	5
Quit the game	6
Display Board.....	7
Data structure Choice.....	8
Ending of game	9
Error Handling	10
Additional features.....	11
Undo/Redo Moves	11
Solver	14
Hint Feature.....	15
Additional Information	16

Building the game

The following steps are involved in building the project:-

1. Open terminal in the directory in which the Haskell file is present in
2. Type `ghc -o FinalProject 3035341922.hs` to generate the executable
3. Type `FinalProject` to start the executable

```
C:\Users\subha\Downloads>ghc -o FinalProject 3035341922.hs
[1 of 1] Compiling Main             ( 3035341922.hs, 3035341922.o )
Linking FinalProject.exe ...

C:\Users\subha\Downloads>FinalProject
Welcome to Jigsaw Sudoku
The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :
```

Game Interface

The game interface has been made user-friendly by assigning a digit to each option available. The user has to only input the digit associated with the option desired.

```
The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :1
```

Basic Functionalities

Load a file

The game interface requires the user to just enter the digit related to the option. Type “1” to load the board present in the text file.

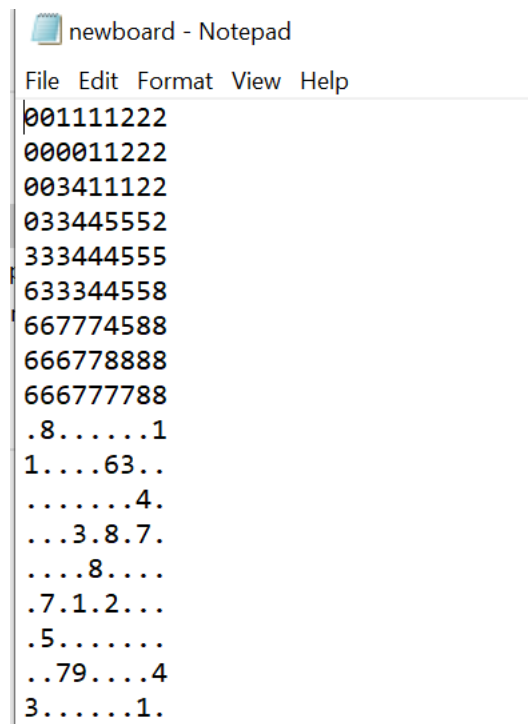
```
My choice is :1
Enter the file name:sudoku.txt

Here is the board
+-----+
| . 8 | . . . | . . 1 | |
| 1 . | . . . | . 6 3 | . . |
| . . | . . . | . . . | 4 . |
| . . | . 3 . | 8 . 7 | . . |
| . . | . . 8 | . . . | . . |
| . 7 | . 1 . | . 2 . | . . |
| . 5 | . . . | . . . | . . |
| . . 7 | 9 . | . . . | . 4 |
| 3 . . | . . . | . . 1 | . . |
+-----+
```

Save a file

For saving a file the user has to just enter the digit “2” which is associated with saving a file among the options available. The user has to then enter the filename of the board which will be saved in the same directory as the Haskell file. The file format is the same as the input. First, we have the blocks of the cells followed by the values.

```
The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :2
Enter the file name:newboard.txt
```



```
newboard - Notepad
File Edit Format View Help
001111222
000011222
003411122
033445552
333444555
633344558
667774588
666778888
666777788
.8.....1
1....63..
.....4.
...3.8.7.
....8....
.7.1.2...
.5.....
..79....4
3.....1.
```

Make a move

To make a move the user must type “4” in the Options Panel. Once the user has typed “4”, there will be a prompt to enter the row, column and the number. Row and Columns are indexed from 0 and hence the range allowed is 0-8. The numbers allowed are 1-9.

```

The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :4

Next move:
Enter Row number:
0
Enter Column number:
0
Enter the number:
2

```

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

Quit the game

For quitting the game, the user has to type “5” and this will quit the game and display the message “Thank you for playing Jigsaw Sudoku”.

```
The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :5
Thank you for playing Jigsaw Sudoku
*Main>
```

Display Board

For displaying a board, the user has to just type “3” in the options Panel. An illustration of the current state of the board will appear on the command prompt.

```

The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :3

```

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

Data structure Choice

```
type Sudoku= [(Int,Char)]
```

The Jigsaw Sudoku board was represented using a list of type having an Integer and Character. The Integer represents the box/block number of the cell and the character is either '.' or the number from 1-9. A list is used since there will be 81 cells for a 9x9 board. We can use the !! operation to get a cell since they are indexed from 0-80. This data structure has been chosen for its simplicity since it allows us to store the key information of the box/block number and the content of the cell and allows for direct access for that cell. We can also directly obtain rows of the board by using the "take" operation and also obtain the column using the "mod" operation. By searching through the list, we can also obtain the boxes/blocks within the board. This would ensure efficient implementation for the remaining tasks.

Ending of game

The game will be terminated when the user opts to quit the game by typing "5" in the options panel. The game will also terminate and display a congratulatory message when the user completes the game which is automatically detected by the program.

```
The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :5
Thank you for playing Jigsaw Sudoku
```

```

The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice and press enter
My choice is :4

Next move:
Enter Row number:
8
Enter Column number:
4
Enter the number:
2
Congratulations you win the game

```

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

Error Handling

The following error handling conditions have been ensured:-

- i) The program ensures that a board has been loaded before the user can opt to choose any of the other options i.e. the user has to type "1" load board before moving to other options

- ii) If the user enters anything apart from 1-9 in the options, the program detects and prompts the user “Invalid option entry” and asks to re-try the by displaying the options panel again
- iii) Numbers are checked to ensure they are within 1-9 and Row and Columns numbers are checked to ensure whether they are between 0-8 when a move is made. If a valid number is entered for row number, column number or value of cell the user is prompted that “number is out of index” and asked to re-try by displaying the options panel
- iv) If the user opts to redo or undo a move when no move has been made, then a prompt is displayed “No moves to undo/redo” and user is asked to re-try by displaying the options panel again
- v) The program always checks if the cell is empty before the user can fill in a value. If a cell is not empty, then a prompt is displayed “Invalid move” asking the user to re-try by displaying the options panel
- vi) Sudoku conditions of row, column and box/block duplication are checked
- vii) Filenames and input file content are assumed to be valid

Additional features

Undo/Redo Moves

The program has been developed such that the user can undo/redo their previous moves provided it meets validation conditions of the latest board state.

For efficient implementation for this operation I have created a custom datatype called Move of the type [(Int,Char)] where the Int stores the index from 0-80 and the Char stores the number. Every time a valid move is made, the move is added to the front of a list called “moves”. When a user wants to undo a move, the move from the front of this list is popped and pushed to the front of a list called “undone” restoring the previous state of the board. When a user wants to redo a move, the move from the front of “undone” is popped and pushed to the front of “moves” restoring the previous state of the board. This is exactly like two stacks which have been implemented using lists in Haskell thereby not using any use data structure.

Initial State of Board

Here is the board

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

Valid move made to place "2" at row "2" and column "0"

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

Undo operation: The number "2" is removed from Row "2" and Column "0"

```

The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :7
Un-doing your move

```

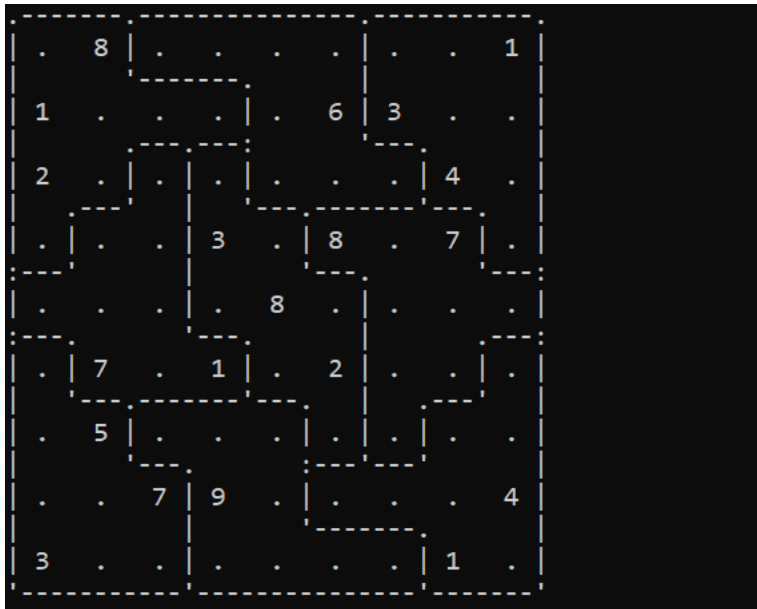
.	8	1
1	6	3	.	.
.	4	.
.	.	.	3	.	8	.	7	.
.	.	.	.	8
.	7	.	1	.	2	.	.	.
.	5
.	.	7	9	4
3	1	.

Redo Operation: The number "2" is restored at row "2" and column "0"

```

The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :8
Re-doing your move

```



Solver

The program has a solver functionality which gives a solution to the current existing board if one exists. The logic involves trying every possible valid number in each empty cell and backtracking to each combination until a combination of numbers exists that satisfies the current board configuration. The backtracking logic involves recurring through all possible configurations giving the first correct configuration as an answer if one exists. The user has to just type “6” and the prompt will present a solution if one exists and terminate the program.

```

The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :6

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

This is the solution
*Main>

```

Hint Feature

The hint feature gives the user a correct valid move that exists in the solution of the current configuration of the board. This feature uses the solver functionality. Once the user opts for an hint by typing "9", the existing state of the board is sent to the solver function and if a solution exists the first number associated with the first empty cell is given as a hint with the exact row and column. If no solution exists, then "No correct hint for this configuration exists" meaning that the user had made an earlier wrong move.

```
The available options are:
1. Load Board
2. Save Board
3. Display Board
4. Make my move
5. Quit
6. Solve board
7. Undo my move
8. Redo my move
9. Need a hint
What would you like to do?
Please enter the integer corresponding to your choice
My choice is :9
Move to Row:0, Column:0 and put number:6
```

Additional Information

1. Although I was unable to implement the correct version of “Board generation” additional feature, I wished to share my approach. The key feature of jigsaw sudoku is to ensure that the blocks are irregular in shape but are connected to each other. In order to do this, my initial approach was to use depth-first search since there would ensure randomness and connectedness. Depth-first search could be executed n times each traversing n unvisited cells each time thereby ensuring that we get n random blocks. However, the complexity I faced in this approach was to ensure that there is no “hole” in the middle (if a cell is picked in the center”). The second approach was to start with a normal sudoku board and then swap boundary cells with neighboring blocks. This could ensure connectedness and avoid the issue of “hole” in the middle, but this would lead to limited number of boards.
2. Although the code currently works for only 9x9 boards, it can be made modular by just changing each of the digits to expressions (eg 81 is $n \times n$, 80 is $n \times n - 1$, 9 is n).