**Final Project**

EECS 368

Arjan Gupta

December 10, 2015

**Project Introduction**

In this project, I aimed to demonstrate my knowledge of the JavaScript programming language. My vehicle to do this was a Sudoku Solver, which I have written using Professor Andrew Gill’s chalk library.

**Table of Contents**

Description of Implementation …………………………………………………………......Page 2

Obstacles Encountered …………………………………………………………………...…Page 3

Lessons learned……………….………………………...…………………………………..Page 3

Resources..……….………………………………………………………………………….Page 4

Screenshots………………………………………………………………………………......Page 5

Code……………………………………………………………………………………...Appended

**Description of Implementation**

Methodology used to write and test program

I started writing this program my using a simple Notepad++ text editor on my Windows 8.1 laptop. I used the chalk.2015.1006.js file provided by Professor Andrew Gill in class as well as a Sudoku.js file created by me in an index.html file (the format for which was also provided by Professor Gill). The program was run and tested using Google Chrome and its Developer Tools Console (by pressing Ctrl+Shift+I).

High-level description of Sudoku.js

The Sudoku.js I wrote is a well-commented file which can be found at the end of this report. The main function of the program calls a ‘pzzl’ function, which has a list of example unsolved Sudoku puzzles. I have also made sure to include the web-page source for each of these unsolved puzzles, as well as each of their unique solutions (also included in the Resources section of this project), but I am confident that this Sudoku solver can solve any unsolved puzzle which is known to have a unique solution. My solver cannot handle puzzles for which there are multiple possible solutions.

After an example puzzle is chosen from the pzzl function (which must be done by opening the Sudoku.js file and changing the argument of the pzzl function in the code), the main function proceeds to welcome the user to the program and displays the Sudoku that is going to be solved. It then says tells the user the Sudoku is being solved. The user can see the program running in Developer’s Console, if they wish to. The harder puzzles can take the solver longer to solve, so sometimes it is entertaining to watch the program do work.

The ‘solver’ function called by the main actually does all the solving work. In the solver function, there is a main while loop that visits all the empty positions (which are marked as zeros in the puzzle displayed), and then checks if values 1 through 9 can be placed in that empty positions in the puzzle. The program places the first value that fits in the empty spot. If no numbers 1-9 fit in the spot, the program goes back to the last empty spot (this is where the back-tracking comes in!), and tries the next possible value. This algorithm is repeated over and over until the program finds a unique solution to fill all the empty spots in the puzzle (I looked up a pseudocode sample on Wikipedia to help me with this, refer to Resources). Finally, the solution of the puzzle is printed out.

**Obstacles encountered**

Different languages were tried initially

At first, this program was attempted in Scheme and Haskell, in that order. I came to realize that the problem was rather challenging in either of these languages just for the simple fact that it is impossible to change values in Haskell and Scheme. The problem could probably be addressed by using ‘set!’ operator in the Scheme (Racket) language, but that is not the way to do things in Racket. In fact, now that I have come to think of it, JavaScript is an amazing language for Sudoku because of its ability to change the size of arrays. However, my determination to solve this problem in a functional language has not been defeated, I aim to explore Haskell over my free time and learn a way to solve this problem in that language. I will also try to write a Sudoku solver in Python.

Major (minor?) bug was encountered

One very frustrating problem I ran into was running into an infinite loop issue. I spent many days trying to debug the issue, only to find that I was not supposed to be using for loops. The moment I implemented while loops instead, my program worked beautifully.

**Lessons Learned**

1. I must never use for loops if I intent to manipulate loop indices.
2. JavaScript is more useful than I had thought.
3. Functional languages are harder than I had thought.

**Resources**

Professor Andrew Gill’s chalk.20151006.js library file. *EECS 368, Fall 2015*. Course website obtained from: https://piazza.com/ku/fall2015/eecs368/resources

Daily Sudoku Puzzles. *Sudoku Snake.* www.sudokusnake.com/archive.php. Puzzles used in the code were obtain from:

http://www.sudokusnake.com/dailies/Easy\_10\_17\_09.pdf (Easy puzzle)

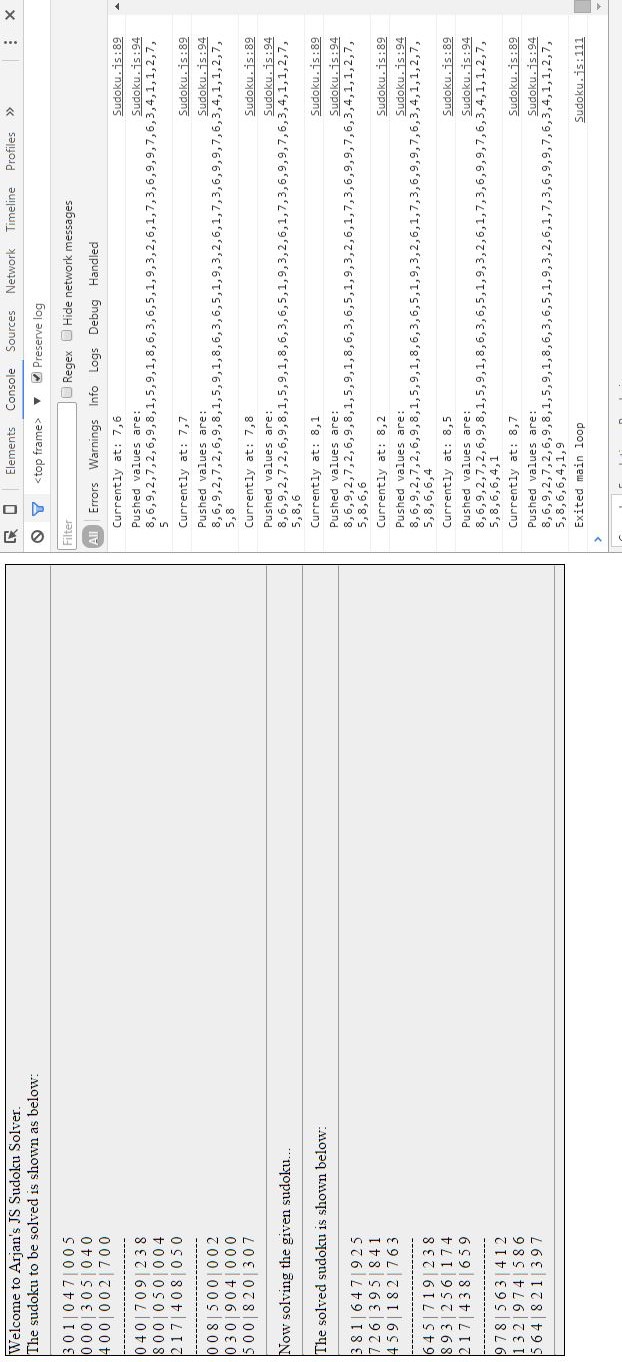
http://www.sudokusnake.com/dailies/Fiendish\_10\_17\_09.pdf (Fiendish puzzle)

http://www.sudokusnake.com/dailies/Serpentine\_10\_17\_09.pdf (Serpentine puzzle)

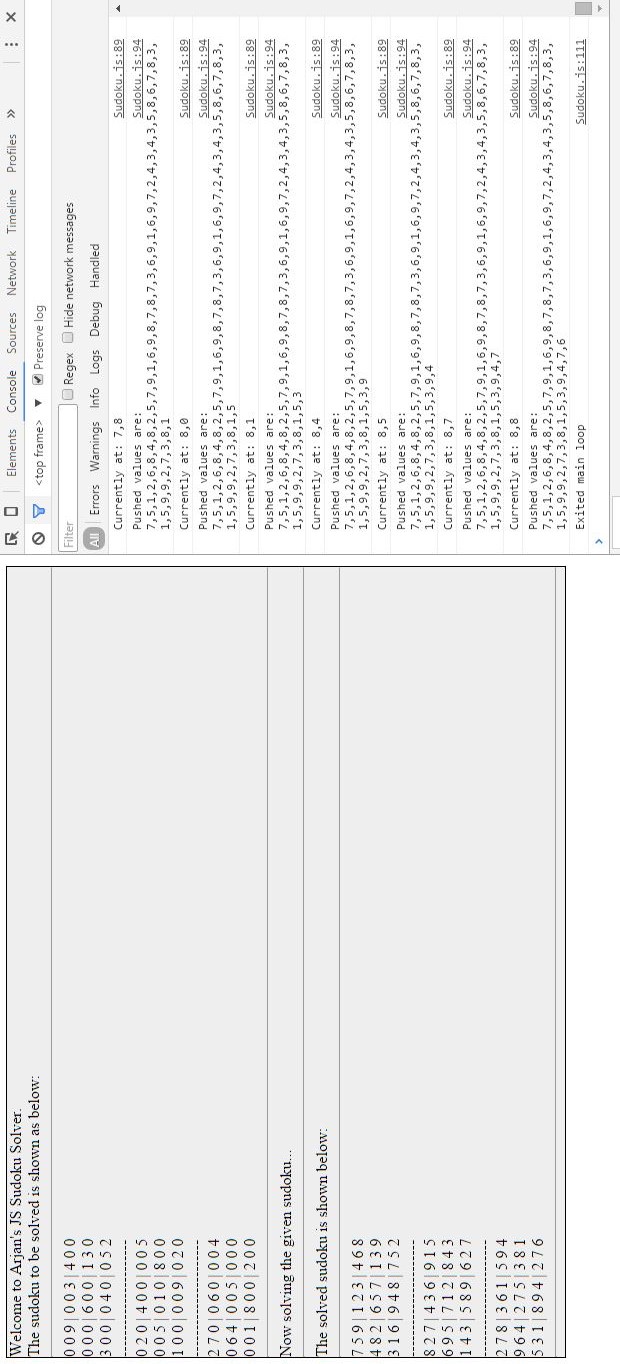
http://www.sudokusnake.com/dailies/Ludicrous\_10\_09.pdf (Ludicrous puzzle)

Backtracking algorithm pseudocode. *Wikipedia, The Free Encyclopedia*. Obtained from: https://en.wikipedia.org/wiki/Sudoku\_solving\_algorithms#Backtracking

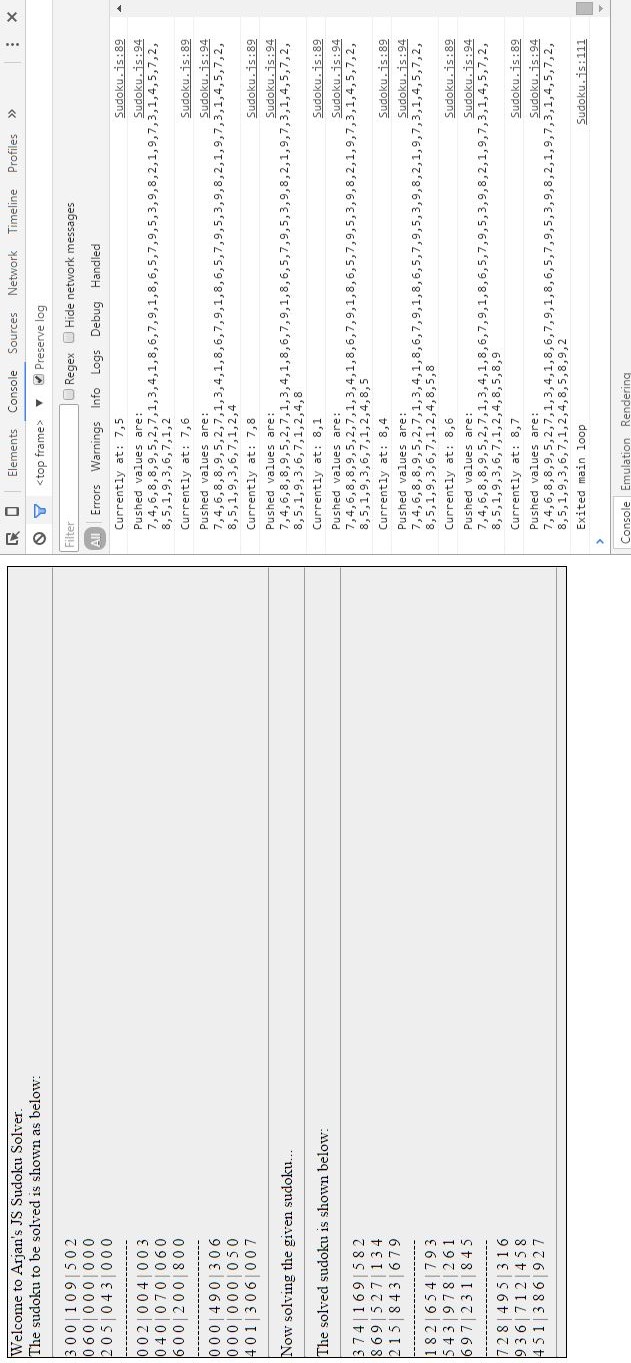
**Screenshots** – Easy Puzzle

****

‘Fiendish Puzzle’



‘Serpentine’ Puzzle



‘Ludicrous’ puzzle

