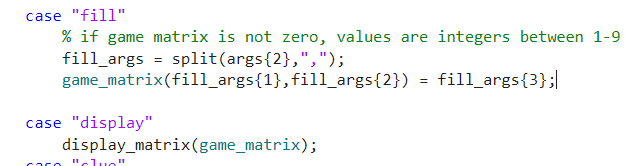
**This document is a compilation of all interesting bugs (excludes syntax errors) that were found in testing during incremental development.**

**Problematic Test 1:**



**Input:**

fill 2,3,4

display

**expected output:**

a 9x9 matrix of zeros with 4 at (2,3)

**actual output:**

a 50x51 matrix of zeros with 52 at (50,51)

**Issue:**

MATLAB was reading in the ascii values of the integers.

**Solution:**

Concatenate the string-form integers to a working format through str2double:

game\_matrix(str2double(fill\_args{1}),str2double(fill\_args{2})) = str2double(fill\_args{3});

**Problematic Test 2:**

|  |
| --- |
| case "fill"  % if game matrix is not zero, values are integers between 1-9  try  fill\_args = split(args{2},",");  catch  fprintf("Invalid input entered\n");  end  % if is integer and between 0/1 and 9  if (~mod(str2double(fill\_args{1}), 1) == 1) && (1<=str2double(fill\_args{1}) && str2double(fill\_args{1})<=9) && (~mod(str2double(fill\_args{2}), 1) == 1) && (1<=str2double(fill\_args{2}) && str2double(fill\_args{2})<=9) && (~mod(str2double(fill\_args{3}), 1) == 1) && (0<=str2double(fill\_args{3}) && str2double(fill\_args{3})<=9)  game\_matrix(str2double(fill\_args{1}),str2double(fill\_args{2})) = str2double(fill\_args{3});  clc  display\_matrix(game\_matrix);  else  fprintf("Invalid input entered\n");  end |

**Input:**

fill

**expected output:**

Invalid input entered\n

**actual output:**

error

**Issue:**

The if statement is throwing errors because it is still executed if the try statement catches an error and compares values that aren’t defined.

**Solution:**

Put the if statement in the try loop too.

**Problematic Test 3:**

|  |
| --- |
| case "explain"  clc  % This explanation is sourced from online.  explanation = "In Sudoku, the objective is to fill a 9×9 grid with numbers from 1 to 9, ensuring that no number is repeated in any row, column, or 3×3 subgrid. This means that each row, column, and subgrid must contain all the numbers from 1 to 9 exactly once. A basic strategy for solving Sudoku puzzles is to start by identifying any numbers that can only appear in one position in a row, column, or subgrid. This can be done by looking for numbers that are already present in a row, column, or subgrid and then identifying the missing numbers. If there is only one missing number in a particular row, column, or subgrid, it must go in the remaining empty cell. This process of elimination helps to gradually fill in more cells and find the solution.";  explanation = textwrap(explanation,40);  fprintf("%s", explanation); |

**Input:**

explain

**expected output:**

The explanation

**actual output:**

error

**Issue:**

Parsing a string to textwrap() when a cell array should have been parsed.

**Solution:**

Use cell array syntax properly.

|  |
| --- |
| explanation = textwrap({explanation}, 40);  fprintf("%s\n", explanation{:}); |

**Problematic Test 4:**

|  |
| --- |
| case "load"  try  if length(args{2}) == 81  is\_valid = 1;  for i = 1:81  if ~isnumeric(args{2}(i)) %comment on this is in the .m file  is\_valid = 0;  break  end  end  if is\_valid == 1  game\_matrix = reshape(args{2},9,9);  clc  fprintf("Successfully loaded the puzzle.\n");  display\_matrix(game\_matrix);  else  fprintf("Invalid puzzle provided.\n")  end  else  fprintf("Invalid puzzle provided.\n")  end  catch  fprintf("Invalid puzzle provided.\n")  end |

**Input:**

load 005300000800000020070010500400005300010070006003200080060500009004000030000009700

**expected output:**

Successfully loaded the puzzle.

**actual output:**

Error

**Issue:**

After a fair bit of debugging, an fprintf("%.0f", args{2}(i)) statement at the beginning of the for-loop discovered that args{2}(1) is 48 and not 0 (args{2} being the second argument in the input).

Therefore, the issue is that MATLAB is using the ascii value of zero.

**Solution:**

Concatenate:

|  |
| --- |
| ~isnumeric(str2double(args{2}(i))) |

**Problematic Test 5:**

|  |
| --- |
| % testing to see if the distribution of rows is even  counts = zeros(1,9);  countscol = zeros(1,9);  for i = 1:100  rng("shuffle");  cells = randperm(81,9);  for i = 1:9  row = ceil(cells(i)/9);  col = mod(cells(i),9);  counts(row) = counts(row) + 1;  countscol(col) = countscol(col) + 1;  end  end  disp(counts)  disp(countscol) |

**expected output:**

Two 2D arrays

**actual output:**

Error

**Issue:**

Modulus in this case returns a value from 0-8, but a value from 1-9 is necessary.

**Solution:**

Add 1 to the modulus

|  |
| --- |
| col = mod(cells(i),9)+1; |

**Problematic Test 6:**

|  |
| --- |
| function [selected\_matrix] = select\_random\_matrix(difficulty)  if strcmp(difficulty, "easy")  filename = "easy\_puzzles.csv";  % prevents using another if statement later  % (fixes grammar for print statement)  difficulty = "n easy";  elseif strcmp(difficulty, "hard")  filename = "hard\_puzzles.csv";  end  matrices = csvread(filename);  rng("shuffle");  random\_selection = randi(height(matrices));  matrix = zeros(1,81);  for i = 1:81  matrix(i) = matrices(random\_selection,i);  end  selected\_matrix = reshape(matrix,9,9);  clc  fprintf("Successfully loaded a %s puzzle.\n", difficulty);  end |

**Input:**

select\_puzzle easy (function argument is “easy”)

**expected output:**

a randomly selected Sudoku game (9x9 matrix) from a list of Sudoku games.

**actual output:**

Error

**Issue:**

The file extensions of the puzzle lists are defined as csv but in reality are txt, as I forgot to change it when I was converting the lists to csv format (to try and incorporate all learnings from practicals, etc)

**Solution:**

Change the actual filenames from easy\_puzzles.txt and hard\_puzzles.txt to easy\_puzzles.csv and hard\_puzzles.csv

**Problematic Test 7:**

|  |
| --- |
| (the entire solve function) |

**Input:**

load 000000002060300570800004000050100007900000000006700030100003200020500080000090004

(a very hard puzzle)

Next input:

solve

**expected output:**

(The solution, as I have verified that it is solvable by other means)

**actual output:**

“This puzzle is invalid, no solutions were found.”

**Issue:**

The solve function doesn’t recurse enough to solve this puzzle and caps out at 1,500,000.

**Solution:**

Change wording from “is invalid” to “is likely invalid” and, in the instructions add a note saying that the solver has limitations.

**Problematic Test 8:**

|  |
| --- |
| case "imshow"  % scale to values between 0-255  scaled\_matrix = (game\_matrix .\* (255/9));  disp(scaled\_matrix)  % show image  imshow(scaled\_matrix); |

**Input:**

extras imshow

**expected output:**

greyscale image

**actual output:**

solid black/white image

**Issue:**

Imshow is not realising that it needs to display the matrix in greyscale.

**Solution:**

Change wording from “is invalid” to “is likely invalid” and, in the instructions add a note saying that the solver has limitations.

**Problematic Test 9:**

|  |
| --- |
| case "undo"  game\_matrix(last\_edit(1),last\_edit(2)) = last\_edit(3);  fprintf("Undid the previous move.\n"); |

**Input:**

undo (before entering a move)

**expected output:**

“Undid the previous move.\n” (which is defined as zero)

**actual output:**

error

**Issue:**

last\_edit was initialised as [0,0,0] instead of [1,1,0] resulting in assigning to a non-existent element.

**Solution:**

Initialise last\_edit as [1,1,0].

**Problematic Test 10:**

|  |
| --- |
| case "undo"    game\_matrix(last\_edit(1),last\_edit(2)) = last\_edit(3);  clc  display\_matrix(game\_matrix);  fprintf("Undid the previous move.\n"); |

**Input:**

puzzle easy

fill 1,2,3

undo

**expected output:**

“Undid the previous move.\n” and the resultant 9x9 puzzle

**actual output:**

“Undid the previous move.\n” and an arbitrarily large (>20x20) puzzle.

**Issue:**

last\_edit was the integer 12 and not a 1x3 array. This is due to elements in last\_edit being assigned with un-concatenated integers.

**Solution:**

Concatenate the ingegers.

**Problematic Test 11:**

|  |
| --- |
| case "randword"  dictionary = fileread("dictionary\_of\_techniques.txt");  dictionary = splitlines(dictionary);  rng("shuffle");  rand\_index = randi(length(dictionary));  fprintf("%s\n", dictionary(rand\_index)); |

**Input:**

extras randword

**expected output:**

a random word from the specified dictionary.

**actual output:**

error

**Issue:**

“fprintf Function is not defined for 'cell' inputs.”

**Solution:**

Use the {} brackets to handle cell arrays.

|  |
| --- |
| fprintf("%s\n", dictionary{rand\_index}); |