P8H - ML Functions

In your MATLAB assignments, we have covered the following:

- Access MATLAB, create an M-File, and submit that M-File and its output to Canvas
- Develop a basic code from a pseudo-code or flowchart
- Comment your code properly
- Basic Code Debugging
- Understand some basic Built-in Math functions
- Scalars, Arrays, and Matrices
  - Creating them as Variables
  - array & matrix math
  - Managing and indexing
- Creating arrays and use array math
- Input data from the command window or files
- Output data to the command window or a figure
- Relational and Logical Operators
- Conditional Statements (IFTT)
- For Loops

Some Useful Commands:

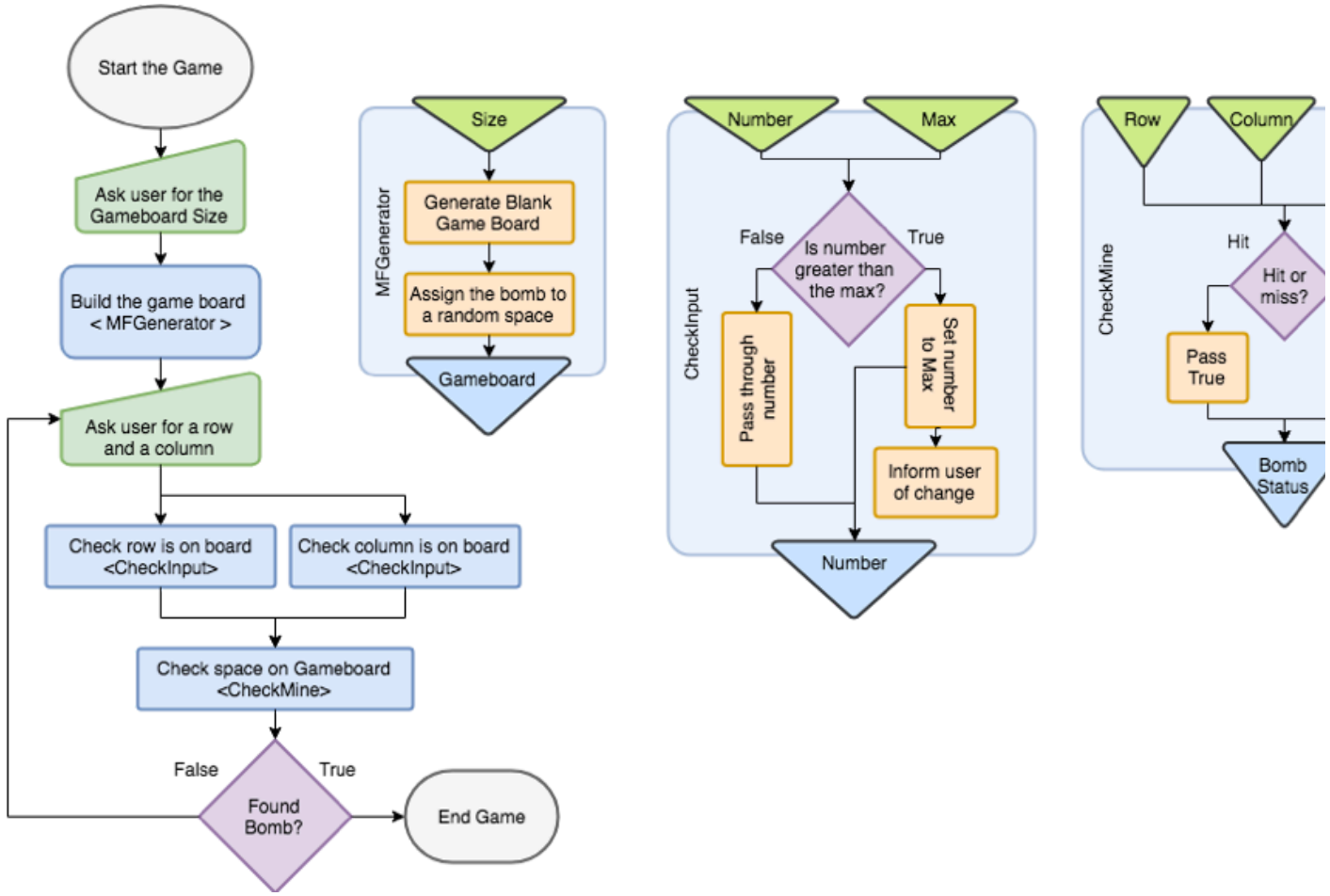
As a reminder, here are some useful commands:

- To comment out a line or a selection of multiple lines
  - PC: Ctrl + r
  - Mac: Cmd + /
- To force a script to end:
  - PC: Ctrl + c or Ctrl + Break
  - Mac: Ctrl + . (period)

Part 1 - Write a Minefield Game

Let's find the mine!

Create a program that follows the flowchart below. It requires 3 functions: *MFGenerator, CheckInput, & CheckMine.* This also shows the expected inputs and outputs of each of the functions.



The function *MFGenerator* is provided to you as an example of a function and to get you started. You will have to write the other 2 function to be used within the game, *CheckInput* and *CheckMine*.

MFGenerator is available on Top Hat or Git Hub:
https://github.com/boconn7782/CourseCode/tree/main/MATLAB/P8H1_Functions
There's also 2 other function in there just for fun. *GameLogo* displays an ASCII game logo at the beginning of the game and *Boom* displays an ASCII message at the end of the game. You don't have to use them, I just thought they made it more interesting.

CheckInput must take the user input for either the row or column (Number) and check that against either the maximum allowable row or column based on the game-board size (Max). It then should

either output the number as is or change its value to the maximum value and output that number.
CheckMine should take those user inputs(Row & Column) and the game-board matrix (Gameboard) as inputs and then determine if the user input is either a hit or a miss.

The game should use a while loop that doesn't break until a hit has occurred.

Here is an examples output:

```
************************************************************

 __  __  _____  _  _  _____  _____  _____  _____  __    _____
(  \/  )(_  _)( \( )( ___)( ___)(_  _)( ___)(  )  (  _ \
 )    (  _)(_  )  (  )__)  )__)  _)(_  )__)  )(__  )(_) )
(_/\/\_)(____)(_)\_)(____)(__)  (____)(____)(____)(____/
************************************************************
Define your gameboard size
Enter number of rows: 4
Enter number of columns: 7
Gameboard size is 4 rows by 7 columns
Let's find the mine!!

Please enter your row:  1
Please enter your column:  1
Miss, Please try again
Please enter your row:  3
Please enter your column:  9
Invalid number. Resetting to max, 7.
...
Please enter your row:  6
Invalid number. Resetting to max, 4.
Please enter your column: 5
It's a hit. You WIN!
   _____      U  ___ u   U  ___ u  __  __     _      _
U | __")u     \/"_ \/     \/"_ \/U|. \/ .|uU|"|u U|"|u
 \|  _ \/     | | | |     | | | |\| |\/| |/\| |/ \| |/
  | |_) |.-,_| |_| |.-,_| |_| | | |  | |  |_|   |_|
  |____/  \_)-\___/  \_)-\___/  |_|  |_|  (_)   (_)
 _|| \\_       \\         \\   <<,-,,-.   |||_  |||_
(__) (__)     (__)       (__)   (./  \.) (__)_)(__)_)
```

NOTE: The heading and footer ASCII images are included as *GameLogo.m* and *Boom.m*. **You do not need to use these if you do not want to.** I just enjoyed these additions a previous student added. Feel free to do something different to start and end your game.

Pseudocode

As always, we start by going over our logic. So let's start with a very loose pseudocode.

We know we need to do the following:

- Create a main program with 3 functions
  - We don't know how to create MATLAB functions yet but we can trust that that'll be part of this lesson.
- Figure out the game-board side from the user and generate the game-board
- Ask the user for a column and row until they find the bomb
- Create 2 functions:
  - One that checks a number against a maximum size
  - One that checks if a game-board space is the bomb

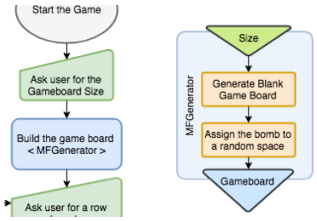
Here's a loosely formatted pseudocode to start with:

https://raw.githubusercontent.com/boconn7782/CourseCode/main/MATLAB/P8H1_pseudocode.txt

We're going to use it as a start but update for format as we go.

Update on Logic Representation

Let's go over how you should represent functions in your logical representations. Now you usually don't want to include them at first, only when you start recognizing patterns or areas where you can offload some complexity to an isolated function. It is clear cut here because of the assignment requirements but I'll walk you through the reasoning.



First, let's talk about how we represent functions. Some of the language will become clear once we go over them. You can see how they're represented in a flowchart from above. You give them their own flowchart with inputs at the top and outputs at the bottom. In the main flowchart, include a process block with an indicator of what function is being called. So essentially you're just making a flowchart to further detail a process in your main program.

What about in pseudocode though? Well you have your main program and then you have functions that your program can call. So we denote our main program with `PROGRAM` and we denote the functions with `FUNCTION`. When we refer to them in the text, we say `CALL <function name>`. You can also use `GET <function output> FROM <function name>`.

And if you need to include inputs for your function, add `WITH <the inputs>` when you're capturing an expected output. In the function pseudo code, you should use `PASS IN <inputs>` and `PASS OUT <outputs>` for handling the information being taken in or sent out.

Let's talk walk through determining when to use a function while showing some pseudo-code examples. First of all, there's never a hard need for a function. You can repeat code through copy-paste and slight edits. You do end up with a long and unwieldy code that's all about the brute force. I am fine with brute force but you can't rely purely on that. Let's take a look at the beginning of the problem, skipping to where you check the user inputs to see if they're on the game-board:

```
1. PROGRAM P8H1:
2. ... After you've asked the user for the gameboard size, built the
   gameboard, and started playing (See flowchart above) ...
3. You know the maximum possible value of rows and of columns
4. Ask user to guess a row and a column value
5. IF the row guessed is above the maximum number of rows
6.     Set row used to the maximum value of rows
7.     Inform user that the program will use the maximum value of rows
   instead of their guess
8. ELSE
9.     Use the row guessed
10. ENDIF
11. IF the column guessed is above the maximum number of columns
12.     Set column used to the maximum value of columns
13.     Inform user that the program will use the maximum value of
   columns instead of their guess
14. ELSE
15.     Use the column guessed
16. ENDIF
17. ...
18. END
```

Notice the similarity between the two If statements. If you replace row or column with X, you end up with the same exact logic. Since we're repeating steps with just different inputs, we can isolate that into its own function. I use `GET` because I want an output from the function and not just `CALL`ing it.

```
1. PROGRAM P8H1:
2. ...
3. You know the maximum possible value of rows and of columns;
```

```
4. Ask user to guess a row and a column value;
5. GET row value FROM CheckInput WITH row guess and max rows;
6. GET column value FROM CheckInput WITH column guess and max columns;
7. ...
8. END
9.
10. FUNCTION CheckInput:
11. PASS IN: Value input and max value;
12. IF value input is greater than max value
13.     Let user know the input is changing;
14.     PASS OUT: max value;
15. ELSE
16.     PASS OUT: value input;
17. ENDIF
18. ENDFN
```

Now in the case of 2 instances of 6 lines of pseudocode, I've reduced it to 2 lines in the main pseudocode and added 11 lines of a function. What if it was a larger block of code and more instances, then you can save yourself even more lines of code. And that's just the pseudocode. Image how much actual programming you saved yourself by recognizing a pattern then accounting for it in your logic.

Functions are not always created for a repeated task though. Sometimes they're done to remove some complexity from the main program and set it aside. That's the reasoning behind including MFGenerator and CheckMine. These are isolated actions of the program. You can easily test them on their own and then set them aside once their working. This takes time to recognize when these might be better set aside in a function. It may even be a decision you make after you've started programming. In this class, I will be specifying when I want functions for isolating a task.

Now back to the

Part 1 Cont'd

Simple as that. let's write it as loosely formatted pseudocode:

```
1. PROGRAM P8H1:
2. Ask user for the max rows and columns for the game-board;
3. GET game-board FROM MFGenerator WITH max rows and max columns;
4. WHILE Bomb not found
```

```
    Ask user to guess a row and a column value;
    GET row value FROM CheckInput WITH row guess and max rows;
    GET column value FROM CheckInput WITH column guess and max
columns;
    GET bomb status FROM CheckMIne WITH row, column, and game-board;
    IF bomb status IS true
        Bomb found
    ELSE
        Bomb not found
ENDWHILE
Inform user that the bomb is found and game is over
END

FUNCTION MFGenerator:
PASS IN: Row and column size for the game-board
Generate a game-board of that size
Randomly assign bomb to a space on that game-board
PASS OUT: Game-board
ENDFN
FUNCTION CheckInput:
PASS IN: Value input and max value
IF value input is greater than max value
    Let user know the input is changing
    PASS OUT: max value
ELSE
    PASS OUT: value input
ENDIF
ENDFN

FUNCTION CheckMine:
PASS IN: Row, Column, and game-field
IF bomb location on game-field matches the row and column input
    PASS OUT: True
Else
    PASS OUT: False
ENDIF
ENDFN
```

This includes the required main program and 3 functions.

Writing the Code

Now let's start the program. Don't worry, there will be a lot of breaks in here to get you over the conceptual hump of creating and calling user-generated functions in MATLAB.

The Main Program

Let's start with the main code. Start an M-File for that and don't forget to:

- Name it properly
- Include a title block at the top
- Include the housekeeping code

Let's use the pseudo-code above to help us start this with some comments.

```
% Provide user information and instructions on the program
% Ask user for the max rows and columns for the game-board;
% GET game-board FROM MFGenerator WITH max rows and max columns;
% WHILE Bomb not found
%     Ask user to guess a row and a column value;
%     GET row value FROM CheckInput WITH row guess and max rows;
%     GET column value FROM CheckInput WITH column guess and max columns;
%     GET bomb status FROM CheckMIne WITH row, column, and game-board;
%     IF bomb status IS true
%         Bomb found
%     ELSE
%         Bomb not found
% ENDWHILE
% Inform user that the bomb is found and game is over
```

The Functions

Let's start with the functions but this will need another break.

Break: Functions in MATLAB

I've included breaks pointing directly to reference information from or similar to the prelab for easy reference as you go.

We've already dealt with functions in MATLAB. You've used `fprintf()` and other built-in functions already. We're going to create our own now. Let's look to the example provided with this assignment. But first:

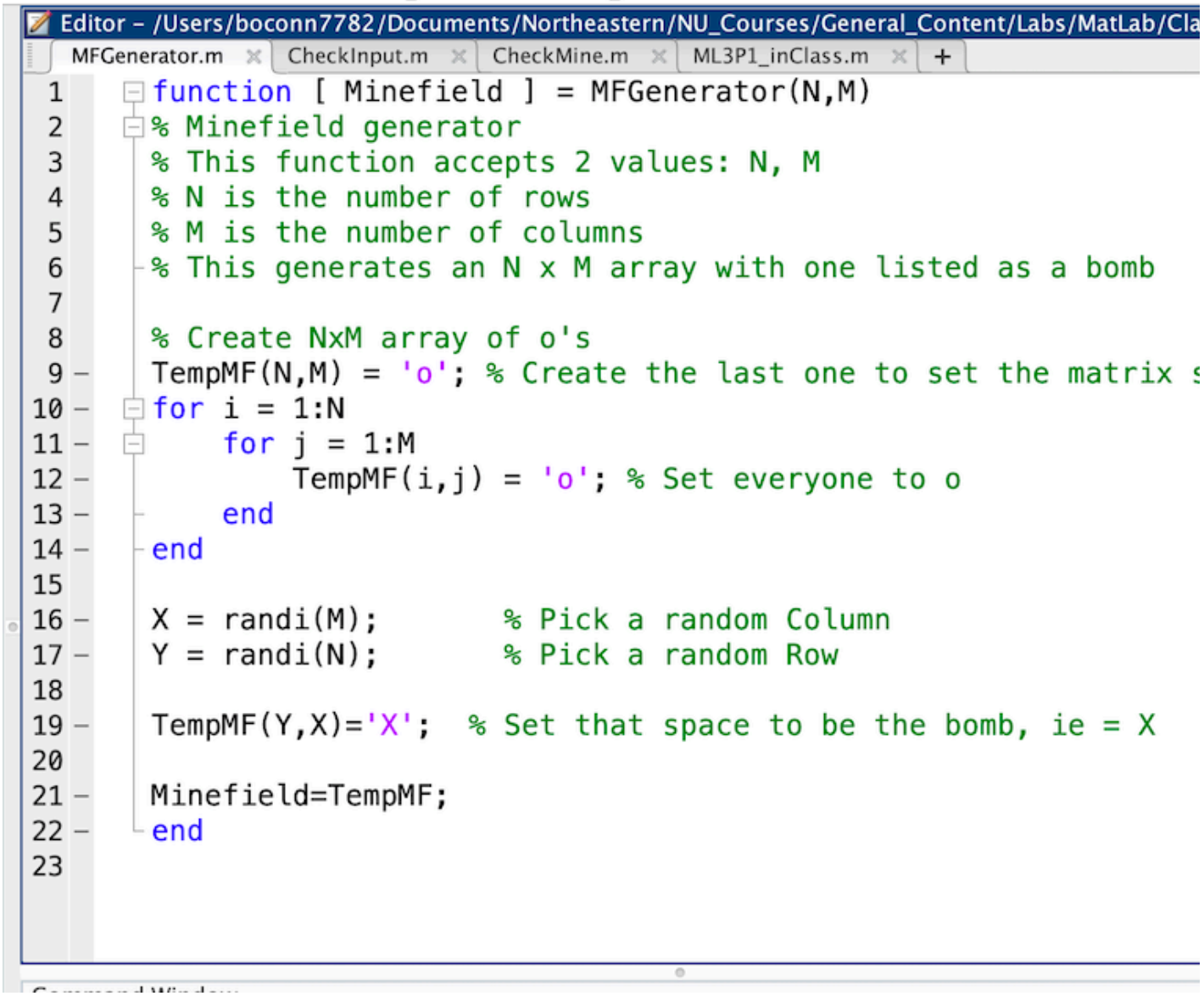
- Download MFGenerator.m, if you haven't already
  - Make sure that it is in your current folder in MATLAB.

You now have a user-generated function at your disposal. Run the following command

NOTE: Run this in your Command Window and not in an M-File that you then run. Use your Command Window for the examples in the MATLAB chapters unless otherwise specified.

1. `MFGenerator(4,7)`

You should see a matrix with 4 rows and 7 columns of `o`'s with 1 of the positions changed to `X`. You've called the function **MFGenerator** with the inputs of 4 and 7. That outputted a matrix to serve as the game-field for the minefield game. Open up *MFGenerator.m*.



Parts of a MATLAB Function:

Let's break this down line-by-line to illustrate how a user-defined function is created:

Function definition line:

The first line defines that it is a function, the outputs, the name of the function, and the inputs. It takes the form `function [y1,...,yN] = myfun(x1,...,xM)`. The first word(`function`) of the M-File will always be function. The array that follows(`[y1,...,yN]`) are the outputs. On the other side of the equal sign (=), the single word is the name of the function(`myfun`) and the name by which you would call the function. The second array(`(x1,...,xM)`) indicates the expected inputs to the function. For *MFGenerator.m*, it shows that the function will output `Minefield` based on the inputs `N` and `M`. To call the function, you must use the name and include the required inputs.

NOTE: When you are saving an M-File as a function, you must name the file the same as the function name you want to use.

Help Text:

The next series of commented lines is the help text. If you run the command `help MFGenerator`, it'll print out in the command window. If you run the command `doc MFGenerator`, it'll open up the MATLAB File Help window with that text in it. You should include some description and instruction for any function you create. Run the following to see how it works:

1. `help MFGenerator`

Function Body(program)

This is lines 8-21 of MFGenerator. This is where you put the inputs to use as part of the actual program that makes up the function. For instance, when we called MFGenerator(4,7) the function assigns N equal to 4 and M equal to 7. If you had an X and Y in your program, you could call MFGenerator(X,Y) and the function would run after assigning N equal to X and Y equal to Y.

Assignment of Values

For the function to be complete, you need to assign a value to all outputs. This is the value that gets passed out of the function to whatever program called it. If you do not assign a value, it will give you an error.

Function definition examples

Function Definition	Outputs	Inputs
function [mpay, tpay] = loan(amt,rate, yrs)	2	3
function [V, S] = SphereVolArea(r)	2	1
function [V, S] = SphereVolArea(r)	1	2
function [A] = RectArea(a,b)	1	2
function A = RectArea(a,b)	1	2
function trajectory(v, h, g)	0	3

NOTE: Notice how when your output is a scalar, you don't need the square brackets. You also don't need to have an output.

Variables in Functions

This is something lots of novice programmers have issue with. When you call functions, you're entering a different domain. You can pass variables in through inputs but even then, you're only passing along their values. The function and the program that calls do not share variables. If you set a variable in one, you cannot just call it in the other.

- Variables are inherently local. They are only known to the script or function where they are created.
- You can declare a variable as global but that is bad practice at this stage. Don't do it.
- You can use the same variable names in your script and in the function but they won't be the same thing.

Example:

```
Function [dee, dum] = tweedle(x,k,c,d)
```

Called in program:

```
[M,N] = tweedle(f,g,h,i);
```

CODING REQUIREMENTS FOR FUNCTIONS:

- **DO NOT** use the same variables in your program and functions.
  - There is nothing stopping you from doing so other than my policy that you do not do this. I require this so that I'm sure you understand that there is a difference between the variables in the function and the main script.
- **DO** include your name and the assignment in the help text.
  - Treat the first few lines of the function help text as your title block for that m-file.

Part 1 The Functions Cont'd

We already have one of the functions we need. It is provided with the assignment and you've already downloaded it. We now need the other two. Create m-files for CheckInput and CheckMine.

We need these to be named specifically for calling them in the program so DO NOT name them based on the standard naming convention.
Now let's use the pseudocode as a scaffold for our function and its comments.

For CheckInput.m :

```
% FUNCTION CheckInput:
% PASS IN: Value input and max value
% IF value input is greater than max value
% Let user know the input is changing
% PASS OUT: max value
% ELSE
% PASS OUT: value input
% ENDIF
% ENDFN
```

For CheckMine.m :

```
% FUNCTION CheckMine:
% PASS IN: Row, Column, and game-field
% IF bomb location on game-field matches the row and column input
% PASS OUT: True
% Else
% PASS OUT: False
% ENDIF
% ENDFN
```

Coding Line by Line

Let's make sure these have the correct first lines to set them up as functions. We know what the inputs are, the information that is passed in, and we know what the outputs are, the information that is passed out. We can either read that in the pseudocode, PASS IN or PASS OUT, or from what we see in the flowchart.

For CheckInput.m :

```
function [V] = CheckInput(Vin, Vmax)
% <Title Block>
% <Description of function>
% IF value input is greater than max value
```

```
% Let user know the input is changing
    V = Vmax;
% ELSE
    V = Vin;
% ENDIF
% ENDFN
```

For CheckMine.m :

```
function [Bomb] = CheckMine (R, C, GF)
% <Title Block>
% <Description of function>
% IF bomb location on game-field matches the row and column input
    Bomb = true; % Equivalent to logical(1)
% Else
    Bomb = false; % Equivalent to logical(0)
% ENDIF
% ENDFN
```

Now these are both based on some IF statements so I'll do one and you can do the other:

For CheckInput.m :

```
function [V] = CheckInput(Vin, Vmax)
% <Title Block>
% <Description of function>
% Check Vin against Vmax
IF Vin > Vmax
    % Change value to be the maximum value and inform the user
    disp('Value over the maximum so changed to maximum');
    V = Vmax;
else
    % Input within acceptable range
    V = Vin;
end
end
```

For CheckMine.m :

```
function [Bomb] = CheckMine (R, C, GF)
% <Title Block>
```

```
% <Description of function>
% IF bomb location on game-field matches the row and column input
<IF Statement for the bomb>
    Bomb = true; % Equivalent to logical(1)
% Else
    Bomb = false; % Equivalent to logical(0)
% ENDIF
% ENDFN
```

Now that we have completed the functions, let's test them by calling them in MATLAB and seeing if they give us the expected results. Run the following in your command window:

```
CheckInput(3,4)
CheckInput(7,4)
```

We should get a 3 in the first case since that's less than 4 and a 4 in the second case since 7 is greater than 4.

```
X = [1,0;0,0]
CheckMine(2,2,X)
CheckMine(1,1,X)
```

We created fake gamefield where we know where the 'bomb' is and ran it in the new function. The first one should provide a 0 since we know the bomb isn't there. The last one should provide a 1 since we know the bomb is in that location.

Now that we have working functions, let's get back to the main MATLAB script. I've started to fill in parts below:

```
<Provide user information and instructions on the program>
% Ask user how many maximum rows and columns for the game field.
Rows = input('How many rows on the game field?');
Cols = input('How many columns on the game field?');
GameField = MFGenerator(Rows,Cols);
% WHILE Bomb not found
% Ask user to guess a row and a column value;
% GET row value FROM CheckInput WITH row guess and max rows;
% GET column value FROM CheckInput WITH column guess and max columns;
% GET bomb status FROM CheckMIne WITH row, column, and game-board;
% IF bomb status IS true
```

```
% Bomb found
% ELSE
% Bomb not found
% ENDWHILE
% Inform user that the bomb is found and game is over
```

Break: While Loops

That's as far as we can go right now since we have one more topic to cover. We've already dealt with For Loops. Now we need to introduce the more open-ended While loop.

```
while expression
    statements
end
```

This loop will repeat as long as the condition in expression is met. If it is never met, then it will continue forever. While loops are useful when you do not know how many times a loop will run or you are not running through a finite process.

To break the loop though, you will have to change that expression to no longer be true. If you do not do that, you can use the command break to end the loop. Break will also end a for loop before its conclusion.

Part 1 Cont'd: Coding Line by Line

So now we have all of the tools we need to fill in this code:

```
<Don't forget your title block>
<Provide user information and instructions on the program>
% Ask user how many maximum rows and columns for the game field.
Rows = input('How many rows on the game field?');
Cols = input('How many columns on the game field?');
GameField = MFGenerator(Rows,Cols);
% WHILE Bomb not found
<start while loop with expression BOMB_FOUND equals 0>
% Ask user to guess a row and a column value;
<Get a row input(r) from user>
```

```
<Get a column input(c) from user>
% Check the user values against maximums
r = CheckInput(r,Rows);
<GET column value(c) FROM CheckInput WITH column guess and max columns;>
% Check Bomb Status
<GET bomb status(BOMB_FOUND) FROM CheckMIne WITH row, column, and game-board>
% IF bomb status IS true THEN end game
< IF statement to change or not change value of BOMB_FOUND>
end
% Inform user that the game is over
<Inform user that the bomb is found and game is over>
```

Use the above scaffolds, and the function scaffolds, to help you finish your program. Remember:

- To call a user-defined function, it must be present in your current window
- Functions require a particular format to their first line
  - `function [y1,...,yN] = myfun(x1,...,xM)`
- There is an example of everything new integrated into the provided scaffolds. Use that to help you develop your code.

END of Part 1

For your **Command Window** output:

- Run your code once with a 2 row by 3 column game-field.
- Play until you win

Part 2 - Largest and Smallest Numbers

READ THE FOLLOWING VERY CAREFULLY

- Pseudocode or flowchart.
  - For this part, you must submit a pseudo-code or flowchart

    - Your pseudo code / flowchart does not have to be perfect.
        - It just needs to be a decent attempt at solving the problem. Your final code may vary somewhat from the pseudocode. If that's the case, simply submit an update with the homework assignment.
        - Use the lesson as an example
- Handwritten flowcharts are acceptable if legible
    - If not, use a word or drawing program to assist
- This is not a group assignment. It is an INDIVIDUAL one.
    - But I do understand if you work together.
    - You must turn in your own code
    - You must acknowledge who helped you in the Title block.
        - If it was another student, name them.
        - If it was a Red Shirt, just say Red Shirts and their first name if you can recall it. The same goes for if you get help from one our TA.
- INCLUDE comments in your code
    - Code is not complete if it is not annotated to help us understand what you are trying to do in your code.
    - Basic rule of thumb is you need enough so that someone with a similar ability level could just look at the comments and fully understand what you're trying to do with your code without having to interpret your actions through the code itself.

Most importantly, the input files provided below will not be the input files the TA uses. The size of the data set will be different but your program should still function. For this part, you are provided a 6x24 matrix as an input file that you will need to sort through. The TA may test your program using a 18x10 matrix, a 24x6 matrix, or any other sized matrix. The point is that you will not know so you must create your program to be capable of taking any size input and meeting the assignment requirements.

Assignment:

Write a program that loads a 6 x 24 matrix of integers from a data file and then outputs the largest and the smallest values for each row onto the console in a full sentence. The input file is available on GitHub

INPUT FILE: `LSNumbers.txt`

Available on Github:
https://raw.githubusercontent.com/boconn7782/CourseCode/main/MATLAB/LSNumbers.txt

Requirements:

- Must use a function, `LSNum`, that takes in an array of values and outputs the min and max.
  - Input: an array of any size (Meaning a 1-dimensional arrray, ie a single row at a time, not the full matrix)
  - output: Minimum value in that array, maximum value in that array
- Do not use the commands `min` or `max`.
  - Develop your own algorithm for determining these within the function
  - If you use `min` or `max`, the penalty is 0 credit for this entire part.
- Load the data in as a matrix and treat it as a matrix
  - Do not break it into individual rows or columns when handling it in the main script. For instance, if the matrix is `X`. Do not use `Y = X(:,1);` then only deal with the array Y going forward. Just use `X(:,1)` when wanting to reference that subset of the matrix. IE, show me that you can understand and handle a matrix rather than needing to handle it as individual arrays, a crutch that came up in homework 2.

Example Output:

(Without a descriptive program introduction):

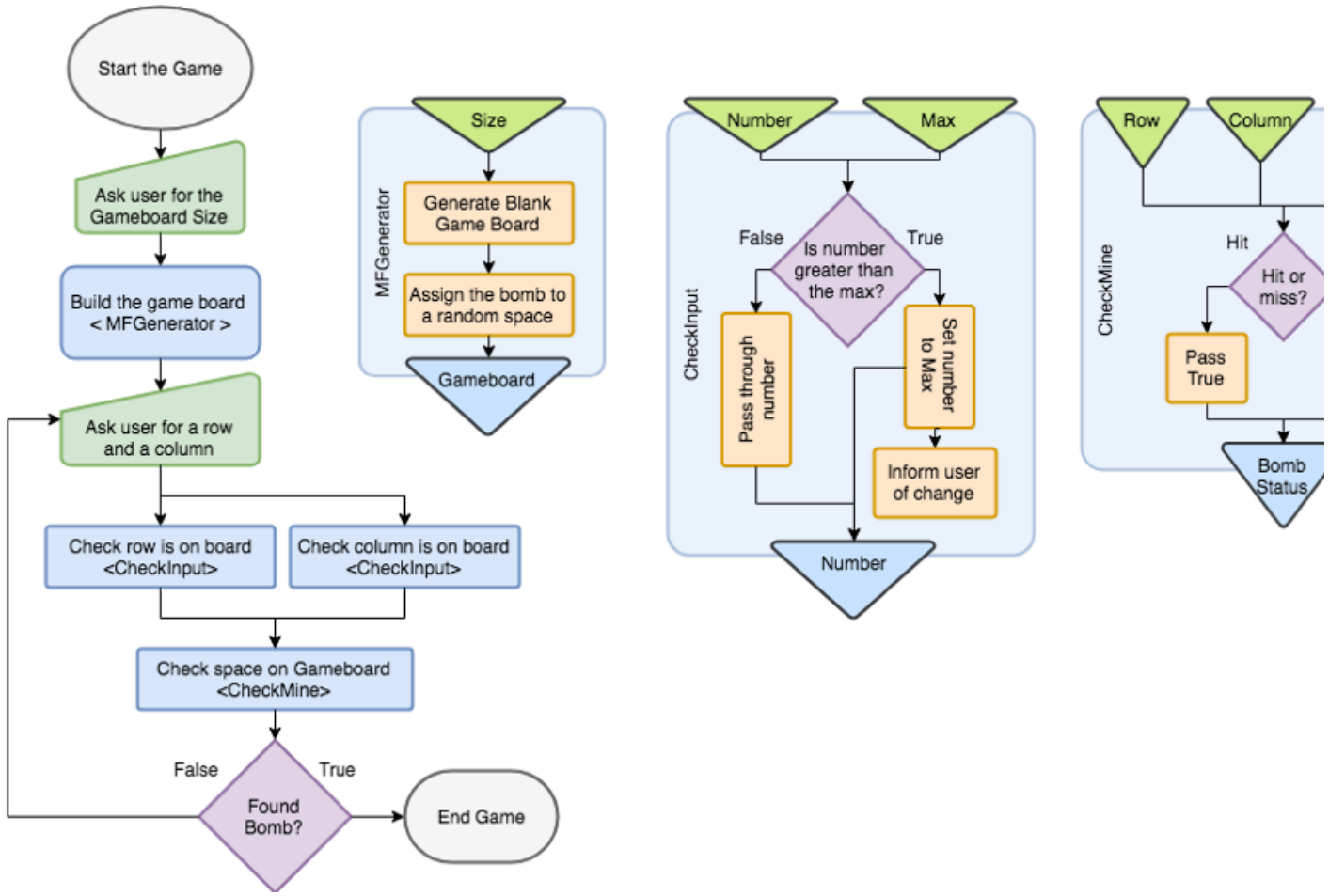
```
********** Largest and Smallest **************
For row 1, the maximum is XXXX and the minimum is XXXX.
For row 2, the maximum is XXXX and the minimum is XXXX.
...
```

This example is incomplete and the numbers are censored out. You will have to completely parse the text file and display the numbers.

Assignment Summary

Part 1 – Write a Minefield Game

Create a program that follows the flowchart below. It requires 3 functions: *MFGenerator, CheckInput, & CheckMine.* This also shows the expected inputs and outputs of each of the functions.



Submission Requirements:

- Use the standard naming convention
  - You should be capable of combining PDFs, or capable of figuring out how (ask at some point), so you'll only have one PDF for each part.
    - If not, you'll have to append a descriptor to the file name.
- For Pseudocode or Flowchart
  - Submit as a .pdf
  - If you are unable to combine PDFs at this point:
    - Save document using a suffix of '_logic' with the standard naming convention.
- For your final code
  - Submit just the M-File

        - You do not need to print your code to pdf. Just submit the M-File
- For Command Window Output
    - Submit as a .pdf
    - If you are unable to combine PDFs at this point:
        - Save document using a suffix of '_Output' with the standard naming convention.

Part 1

Only the Main script should abide by the standard naming convention. The functions should be named *CheckInput.m* and *CheckMine.m*. They should also have the required inputs and outputs, no more. This is so the TAs can run your program based on working functions if there are any issues with yours so they can maximize the partial credit they can provide.

Submit the following:

- P8H1_XXXXX.m
- CheckInput.m
- CheckMine.m
- Command Window Output playing through a game
    - Use a small gamefield to help speed up the game
    - Replay if you find it on the first turn

You do not need to submit *MFGenerator.m*, *GameLogo.m* and *Boom.m.* There should be no changes to these functions so they TA will have the same ones you were provided.

Part 2

- Flowchart or pseudocode
- Command Window Output
- Main script M-File - P8H2_XXXXX.m
- Function M-File - LSNum.m

The following survey is for my reference to help improve future assignments. The results are not checked until after the semester is completed and participation is not required.

Rubric

- -5% for each script or function missing a title block
- -5% for each function missing help text
- -5% for no instructive introduction as part of user interface

P1: Minefinder Game – 50%

- CheckInput.m - 15%
  - CheckInput formatted as required
  - CheckInput performs as required
  - CheckInput's logic clear through comments
- CheckMine.m - 15%
  - CheckMine formatted as required
  - CheckMine performs as required
  - CheckMine's logic clear through comments

- P8H1_XXXXX.m - 20%
  - Main Script formatted as required
  - Main Script performs as required
  - Main Script's logic clear through comments

P2: Largest and Smallest Numbers - 50% Total

- Pseudocode - 20%
  - Logical Approach to problem
  - Well communicated/written
  - Penalties for extreme misuse of course syntax
- LSNum.m - 20%
  - inputs and outputs as required
  - performs as required
  - Massive penalty for using the min() or max() commands (0% for this P2 automatically)
  - Commented throughout and cleanly
- P8H2_XXXXX.m - 10%
  - Clean and clear UI
  - Performs as required