Minesweeper

Developed by: Lai Wei, Mary William,
Matthew Steigauf, Ted Schmitz, and Tran Lu

Abstract:

Minesweeper is a single-player puzzle video game. The objective of this game is to find all mines hidden in a board without triggering them. Our goal is to make Minesweeper using skills we learned this semester via Matlab. Using knowledge obtained from class and help from the internet, the game was created along with additional features such as a tutorial and a leaderboard. Using two fields, a display field which hid the answerfield (which contained all mines and information about them), the game was made so that squares were only revealed once the user uncovered them. Both a text-based and a GUI representation of the game were created. Important elements of the code (other than the GUI) include the zcheck function (which reveals all squares without mines when a user reveals a square containing zero nearby mines), building the answer and display fields, the mechanics of the gameplay loop, the mechanic of flagging squares thought to contain a mine, and the additional features of the tutorial and leaderboard.

Background and Introduction

Since its inception, the Mobile Inhibiting Necro-Enhancer (M.I.N.E., for short) has instilled fear into the hearts of millions. Brave souls called minesweepers would go out in front of their army units to make sure the path was clear to advance. However, in this day and age, this once terrifying profession has been turned into a video game, that entertains all ages. It was the express purpose of our group to bring this game to Matlab.

Minesweeper (the video game) is not some original concept developed by our team. Anybody who has ever worked in an office with a Windows 98 PC, through Windows Vista-era computer, probably remembers this delightful little puzzle game that can waste countless minutes of one's time. For those who were not so privileged, minesweeper is a game in which a player is given a grid containing safe spaces and mines. Their goal is to clear every space that isn't a mine. To do this, a player clicks a square and, if it's not a mine, a number is revealed. This number corresponds to the number of mines present in the adjacent squares. Every time the player makes a move, there is a possibility that there is a mine underneath the unturned spot. This adds to the thrill of the game. The game itself has some special quirks to it, like flagging the mines to help track them or timing the game to win a bet. Through all of it, the core goal is clearing the field without stepping on mines. Though the game is not as stressful as the minesweepers job, it makes up for a good challenge (for us to create as well as the players to play). In this project, we used our experiences from 2401 and created our own version of this iconic game.

Methods:

General Logic: The first thing that needed to be determined was how would the logic of the game functioned on the most macro of levels. How would spaces on the field be known by the program? Would they all be calculated at the beginning and logged or would they be recalculated every time the interface was updated? What would the end user see? Is it the same thing as the program sees? These questions were answered by a quick experiment. We decided to play a game of minesweeper, using a pen and paper, with one person playing the part of the program and giving us the numbers in each revealed square, if we were still alive, or telling us we had died if we hit a mine. This right away made it clear that the most intuitive way of approaching this problem would be to have 2 separate fields, an answer field with all mines and numbers filled in and a display field that would reveal empty spaces to be the corresponding space on the answerfield as the player guessed them.

Field building: Of the two fields, one was very easy. The displayfield only require one value to be displayed by default on every square. This value needed to be impossible to appear in the answerfield, so doing some simple logic the value had to be 9, as 0-8 can appear as values in the answerfield. A set of nested for loops made filling this field a breeze. The answerfield was a bit harder. First, the number of mines in the game needed to be determined through a simple equation, then placed in the field using a random number generator for each coordinate. To fill in the numbers corresponding to the number of adjacent mines to each square, every square would need to be checked. This requires 2 for loops. Then, to check a 3x3 square centered at each point, another set of 2 for loops was required to total up the number of adjacent mines to the subject point. The end result was a (beautiful) quad-nested for loop set. A quick 'if check' was needed to determine if a given point being check was inside of the field range.

Gameplay loop (general, winning/losing): At the most basic level, the game runs as a while loop that runs until the game is either won or lost. Taking user input for size and difficulty, the number of mines are calculated and generated and the field is displayed. The user is prompted to click different squares, akin to the traditional way of playing the game. Clicking a square with no mines around it reveals a zero, along with all adjacent zeroes running up to the nearest mines. In the text-based version of the game, the user can place a "flag" on a square that he or she deems to be a mine. Once the user has revealed all non-mine squares and flagged every mine, the "game won" graphic is displayed along with the score the user achieved. The user is then prompted to enter his or her initials, and if the score is high enough to appear on the leaderboard, both the score and the user's initials will be displayed. In the unfortunate event that the user clicks a mine, the loop is ended and the user is coldly greeted with the "game lost" graphic.

Zcheck: This function is the backbone of the entire game. Without it, a space in the answerfield could not update the displayfield and the user input would not do anything but flag. The major part of this functions functionality comes from it's response to a user asking to check a point that is a 0 on the answerfield (hence the name Zcheck). In Minesweeper, if a 0 is clicked, all other adjacent, uncheck points are check. This continues until the process hits the edge of the field, or a wall of numerical values that are not 0. It has to work in any direction and for any number of checks, as the field can be as big as the user wants. To do this, an if/else set was built with non zero being the else end. If this happened, then the check displayfield point would be equal to the corresponding answerfield point. If the corresponding answerfield point was a 0 however, the points would be set equal to each other, then the Zcheck function would be recursed for all values between x-1 to x+1 and y-1 to y+1. This simple recursive code made an otherwise seemingly impossible task very easy. It should be noted that it is impossible for Zcheck to check a mine. This is because the main code checks for the corresponding answerfield to not be a mine before calling Zcheck, and because all mines are surrounded by other numeric values, Zcheck cannot recurse itself onto a mine. Zcheck also logs a total of every space it clears in a global variable. The main code then checks this variable every loop and if the remaining number of flagged squares is equal to the total number of squares minus the deleted ones, the player wins.

Flagging: For how much of the main code Flagging takes up, it's very simple. First, the player must specify that they want to flag. Then the pick the point. That point on the display field then shows a 17 to represent it being flagged. A player cannot check a flagged space. Every time a flag is placed, the total number of flags on the board (a variable) increases by 1. A flag can be removed by selecting to flag the same spot a flag already is present in. This decreases the total number of flags (variable) by one. To win, all mines must be flagged and all other spaces cleared.

RevealCheck: The reveal check will reveal the grids that logically, according to the display field, are impossible to be a mine. This process is done by comparing the number of target grid to the amount of total flags in a 3*3 square area with the target grid in the center. All the grids revealed by this check will undergo another iteration of **Zcheck** which recursively reveal all the grid marked with 0 until a grid with the number other than zero is encountered. Two *for* loops were used in this check, the first for counting the mines in the 3*3 square area with the target grid in the center, the second one reveals the grids that not been flagged. If there are mines under the unflagged grid, the game ends and the player loss.

Adaptive tutorial for the text-based version: The tutorial explains to the user what numerical value of an element of the grid represents, how to input selection for each move and, most importantly, the logic behind guessing and flagging. This is done by having the user play with a manually generated 5x5 matrix at the easiest level of difficulty (4 mines) as illustrated below.

10	1	1	1	10
1	1	0	1	1
0	0	0	1	1
0	1	1	2	10
0	1	10	2	1

As the user interacts with the tutorial-mode game, for a specific situation that the user may run into, a set of adapted instructions are printed out in the command window if that is the first time the user comes up against such a situation. For example, when the user chooses to flag for the first time, adapted hints on where the user should flag are displayed. For every occasion in which the user reveals a bomb-containing spot, warning messages are shown along with the prompt requesting the user to make a different selection so that the user can never lose in the tutorial mode. The user completes the tutorial once all mines are flagged and the empty spots are revealed.

In total, there are 7 sets of instructions integrated into the body of the "User Interaction/Beginning of Game" section of the main code for the text-based game. Specifically, those sets of instructions are: (1) teaching how to guess, (2) teaching how to flag, (3) teaching how to unflag, (4) explanation for a 0-spot, (5) explanation for 1-spot, (6) explanation for a 2-spot and (7) showing hint to flag if a 0-spot is previously revealed. At the beginning of the tutorial code, 7 corresponding checking values are initialized to be 0. Once a set of instructions have been shown, the respective checking value is set to 1. The if condition statement for executing the display of an instructional set is if(<checkingvalue> == 0).

Leaderboard: The creation of the leaderboard involved two main elements. The first involved the calculation of the final score and how it incorporated the elements that made each game unique and better or worse than others. The second job involved taking the top ten scores (which needed to be remembered between gameplay sessions) and the attached usernames, sorting them from highest to lowest, and displaying them for the user to see.

Creating the scoring equation involved much trial and error. The elements considered when scoring were time taken to win, the size of the grid (squared), and the difficulty level chosen. Since a larger grid and a higher difficulty were analogous with making the game more difficult, they were considered proportional to the final score, with the time being inversely proportional to reward a faster solve. In the end, the final score was calculated as

score = ceil(200(Size)2(Difficulty)log(t)),

where the 200 factor is used to display whole numbers in the thousands, more similar to an arcade-like score. Simple display functions are used to show the user the leaderboard. The scores, along with the user's initials as inputted, appeared in respective vectors s_vect and u_vect, which are sorted by a for and if loop which runs if the score is higher than the score in the loop, using temporary vectors to replace the scores below the new leaderboard entry.

GUI: A Graphics User Interfaces, or GUIs for short, are a convenient way for the end user of a software to enter inputs in your code. When designing a game, it's important to realize people will break it in any way they can, be it from malice or stupidity. A GUI is a good way to simplify what a player can input, and thus control the ways things can go wrong. In the case of Minesweeper and other game, it can improve the experience. Instead of confusing text based displays or counting spaces to enter coordinates, one can simply click where they want to check, simplifying game play and improving enjoyment. However, these positives are only shared on the end user side. From a developing standpoint, I cannot stress enough **DO NOT MAKE A GUI IN MATLAB**. The original text-based game was 4 functions. The GUI, which has less game play complexity, has 7 functions, 3 figures, and 2 associated images. I began work on trying to make some sort of graphics for the game in mid October and only had something functional by the first week of December. The following is a brief explanation on why this set of code is so beefy and difficult.

Let's start in a similar place to where the logic of the text based game began. How will the user and program views of the game differ? This seems simple, the user will have a sheet of buttons to click, that delete themselves on click revealing a number from a sheet of numbers layered behind the button sheet. Then, all we need to do is simply write a code so that the number sheet is filled in with each square on the sheet corresponding to a square on an answerfield and it's ready to play. Except no, no it is not. To begin with, how does each square on the answersheet fill itself in with the correct answerfield space? Each square in the answersheet is an independent StaticTextBox object (just think of objects like a physical function on the GUI that one can position, interact with, and set other properties of) so how will it know where it is on a grid? Let's assume this is solved, the same applies to the buttons. How do they know where they are with respect to each other? What happens when a 0 is clicked? What happens when a 0 is clicked and an already deleted button is called?

These were just some of the first questions faced in the first few weeks of development. Filling in the answersheet was actually quite simple. If a variable was defined to log the number of creation of each object (ie the 2nd created would have a number of 2) then a position would be able to be ascertained. To do this, the size of the grid in number of objects must be known (ie a 5x5 grid). Then, the order the textboxes are created must be known and not random. Lucky for us, we can write the code so that this is known to us. By doing a double nested for loops for 1:size, with going across on the x axis being the outermost loop, the numbers will increase as one starts in the bottom left corner and moves up rows and over columns, ie:

3	6	9
2	5	8
1	4	7

From this, the position of each textbox in this relative SxS square can be found by the equation:

```
posx=mod(number, size);
if posx==0
   posx=size;
end
posy=ceil((number)/size);
```

With the positions of each text box known, their corresponding answerfield values can be filled. The question is how does the program know when to do filling? An object has 3 types of functions associated with it; One called on it's creation, one called on it's deletion, and one called on it's call back (referencing it in another function or clicking it if it's a button). Since the text will not need to be edited once it is written, the on-creation function is used.

At this point it is important to note that 3 features needed to be brought over from the text based function, that being the customizable size, difficulty, and an answerfield. A problem was encountered when trying to do this though. Since the variable S (size) and D (difficulty) would not be defined until the player chooses to, and the button/textbox layout could not be determined until S was defined, another GUI was needed to get these variables from the player. This GUI called BootUp, consisted of a drop down menu for size and difficulty level. It also served some other purposes, like displaying the rules of the GUI version of the game. This GUI also built an SxS field of zeros and defined it as the variable DeletedField, which would be important later. The final thing it did was build the answerfield. This had to be done here, because if it was done in GUI3 (the main GUI) it wouldn't be built before the textboxes began filling. Meaning the answer key and actual answers would not match. The variables S, number of mines, DeletedField, and Answerfield were defined as global variables so all functions had access to them and could edit them without them needing to be outputs/inputs.

The main GUI, GUI3, had to do a few things. The first, was to use the data from BootUp to create the proper sized grid full of buttons and text boxes that would have the proper code to make them function. The next thing was to make sure the buttons would be able to work in a Zcheck style if a 0 was clicked. It also needed to define the global variable of Alive and Victory to be 1 and 0 respectively.

To make a button work, it's position was determined in the same way as the text boxes. This position was then given as a local variable to the button on it's creation along with a function ButtonDown that took it as an input and would run when the button was called back. ButtonDown would first run a function called checkGUI, then see if afterwards the player had either won the game, lost the game, or had to keep playing.

The function checkGUI worked like Zcheck, only with some minor changes. checkGUI took the x-position and y-position of the target button and checked them against the answerfield. If it was a bomb, the global variable Alive was set to false. If it was a non 0, the corresponding space in DeletedField was set to 1. If it was a 0, the corresponding deletedfield was set to 1, and checkGUI was run for the positions at x-1:x+1 and y-1:y+1.

Here's how it all ties together; To begin with, if a player lost, checkGUI would set Alive to false, ButtonDown would register Alive to be False, close GUI3, and open EndGame which would then open the bad ending version. If the player won, ButtonDown would detect this, close GUI3 and open EndGame with the good ending. And if the player had to keep playing, checkGUI would tell ButtonDown that they had not won and were Alive. ButtonDown would then close GUI3 and reopen GUI3. Upon it's reboot, GUI3 would create all the textboxes again with the same answerfield, then it would attempt to create all the buttons, but an if statement would prevent it from creating buttons whose corresponding number/position in Deletedfield was set to 1. Thus, only non deleted buttons would be created, leaving the correct text boxes visible and the remaining buttons still playable. The code never tries to callback deleted buttons, because the only button that is called-back is the one the player clicks. The GUI3 just rebuilds itself between turns.

Some other bonuses were added to the GUI, such as a time function that would display when you win the game so you have something of a score. Another addition was that of color, by which I mean each numerical value that a square could have has its own color based on severity. Lastly, the EndGame GUI has a button to bring you back to BootUp, so you may continue to play without a reboot.

You maybe asking valuable questions such as, "How did you attach variables and functions to the buttons and textboxes,", "How were you able to change the number/position variable for every button/text", or, "For the love of God, why would you do this?!" To answer the last one, because it bothered me that I couldn't do it before. To answer the rest please see the code for more detail but here is the quick version. I would define a string that consisted of the code each object needed minus the customised position stuff. Then I would split the string at the point the position was needed and insert the calculated position value at the break using num2str. Then, I would essentially glue the strings together and add it as a property on the object currently being created.

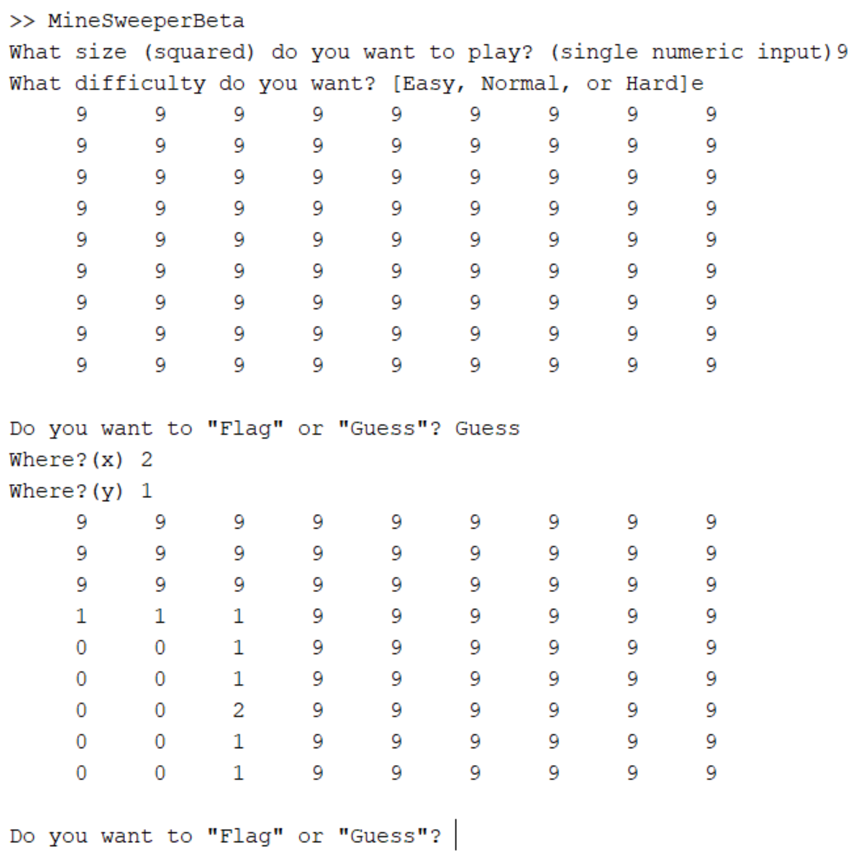
Testing:

The testing section in this project is a complementary piece of code that runs parallel to the text-based game. Many complex parts of the text-based game can be individually evaluated using the code in this section. The code presented below tests for four specific outputs of the text-based code: the difficulty calculation, the location in the answerfeild and what the number represents, and the number of mines Vs difficulty, the dispfeild calculations. The method/ strategy used to write the code was by calling a function from the text based code and comparing its output with that of a new code. By writing the code to generate a desired output, we can test to see if the main function also generates the same output.

Results:

Figure 1: The Text Based Game:

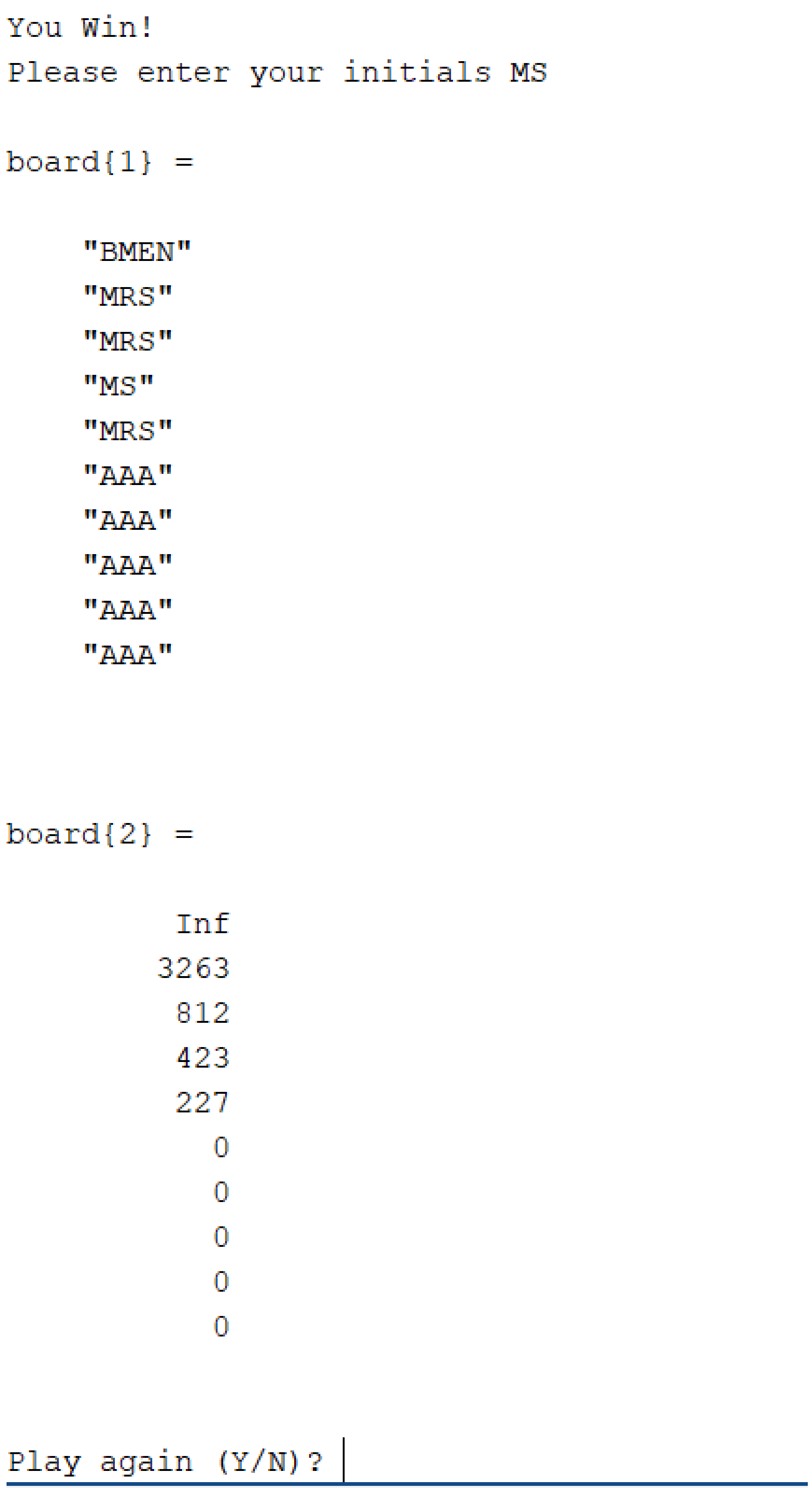
The text version of the game runs in the command window of Matlab, where a user manually enters coordinates to guess mine positions. A 9 represents an unchecked square and numbers 1-8 represent the traditional minesweeper values of mines in direct vicinity.



```
>> MineSweeperBeta
What size (squared) do you want to play? (single numeric input)9
What difficulty do you want? [Easy, Normal, or Hard]e
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9

Do you want to "Flag" or "Guess"? Guess
Where?(x) 2
Where?(y) 1
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     9     9     9     9     9     9     9     9     9
     1     1     1     9     9     9     9     9     9
     0     0     1     9     9     9     9     9     9
     0     0     1     9     9     9     9     9     9
     0     0     2     9     9     9     9     9     9
     0     0     1     9     9     9     9     9     9
     0     0     1     9     9     9     9     9     9

Do you want to "Flag" or "Guess"?
```

Figure 2: The Leaderboard:

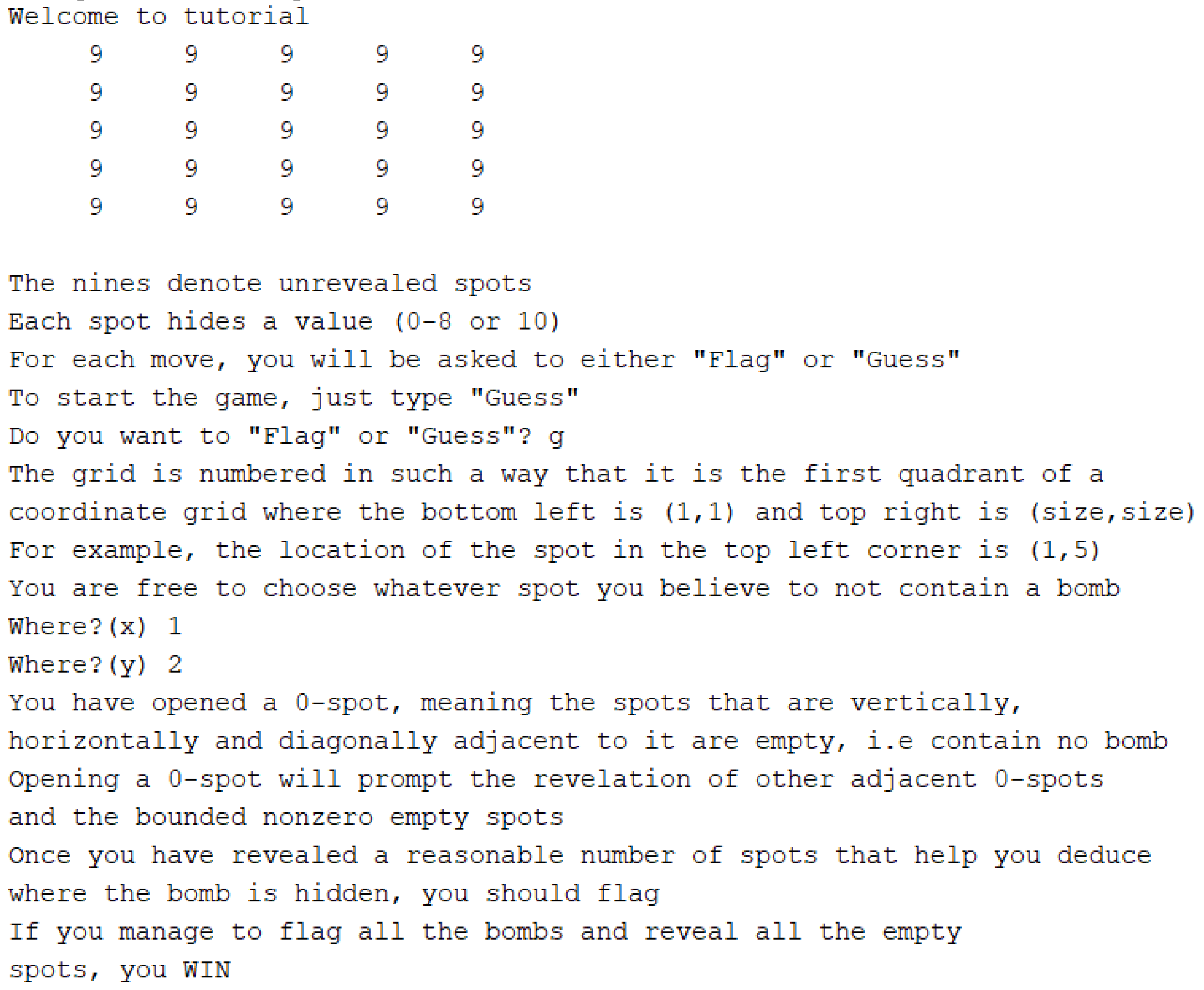
Shown when the user wins the game, the user is prompted to enter his/her initials and the score is shown with the username (not on the same line, however, see the discussion).



```
You Win!
Please enter your initials MS

board{1} =

    "BMEN"
    "MRS"
    "MRS"
    "MS"
    "MRS"
    "AAA"
    "AAA"
    "AAA"
    "AAA"
    "AAA"



board{2} =

         Inf
        3263
         812
         423
         227
           0
           0
           0
           0
           0


Play again (Y/N)?
```

Figure 3: The Tutorial:
An instance of how a tutorial looks like in the command window. The welcome message and the instructions to make the first guess are shown at the beginning of all the instances. In this

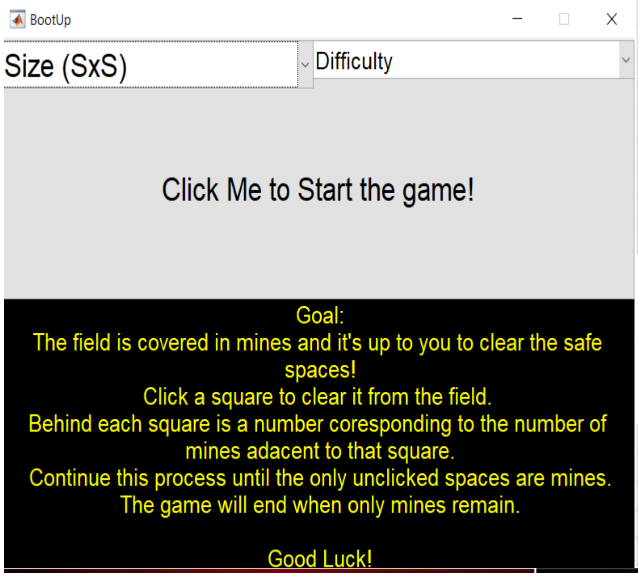
instance, the user opens a 0-spot at their first guess so the next set of instructions to be shown is the explanation of a 0-spot.



```
Welcome to tutorial
     9     9     9     9     9
     9     9     9     9     9
     9     9     9     9     9
     9     9     9     9     9
     9     9     9     9     9

The nines denote unrevealed spots
Each spot hides a value (0-8 or 10)
For each move, you will be asked to either "Flag" or "Guess"
To start the game, just type "Guess"
Do you want to "Flag" or "Guess"? g
The grid is numbered in such a way that it is the first quadrant of a
coordinate grid where the bottom left is (1,1) and top right is (size,size)
For example, the location of the spot in the top left corner is (1,5)
You are free to choose whatever spot you believe to not contain a bomb
Where?(x) 1
Where?(y) 2
You have opened a 0-spot, meaning the spots that are vertically,
horizontally and diagonally adjacent to it are empty, i.e contain no bomb
Opening a 0-spot will prompt the revelation of other adjacent 0-spots
and the bounded nonzero empty spots
Once you have revealed a reasonable number of spots that help you deduce
where the bomb is hidden, you should flag
If you manage to flag all the bombs and reveal all the empty
spots, you WIN
```

Figure 4: The Start-up GUI, Gameplay GUI, and End Screen GUIs:

Shown below is the GUI adaptation of the game, where the user simply clicks the desired square. Pictured from left to right, top to bottom: the start-up screen with size and difficulty adjusters, the game field, and the loss and win graphics.



Click Me to Start the game!

Goal:
The field is covered in mines and it's up to you to clear the safe spaces!
Click a square to clear it from the field.
Behind each square is a number coresponding to the number of mines adacent to that square.
Continue this process until the only unclicked spaces are mines.
The game will end when only mines remain.

Good Luck!

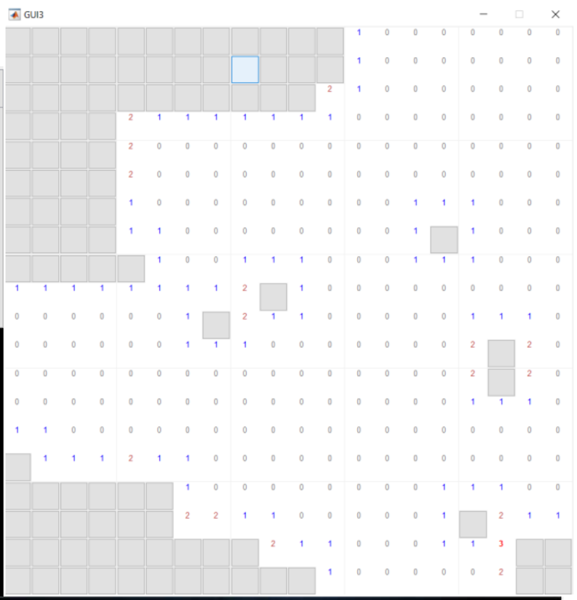


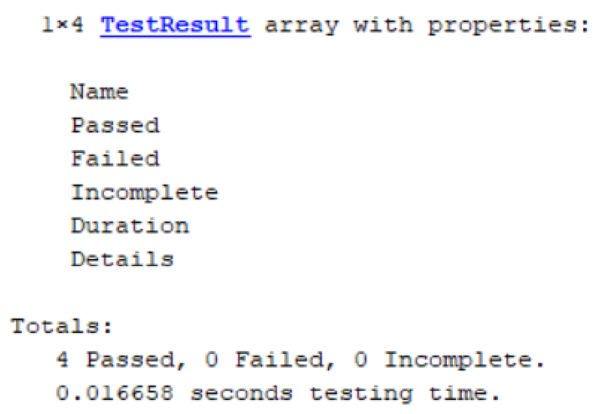
Figure 5: Main Menu:

This menu is the first thing a new user of the project opens. It has a portal to the tutorial, text based game, and GUI game.

```
>> StartTheProject

Hello and Welcome to Minesweeper! Presented by:

                Lai Wei
              Mary William
            Matthew Steigauf
              Ted Schmitz
                  and
                Tran Lu


Game Modes: (1)Turtorial, (2)Text Based, (3)Graphics Based, (4)Close
What game mode do you want to play? Type the corresponding number and press Enter
Game mode 
```

Figure 5: Implementing testing:

This image is generated in the command window after running (testing) the function MineSweeperAlphasTest. This piece of code shows how we can test code in a way that it generates a result window showing us where the code is failing.



```
  1×4 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   4 Passed, 0 Failed, 0 Incomplete.
   0.016658 seconds testing time.
```

Discussion:

Text Based version: The text-based version of Minesweeper presents all the functions and game play logic of the classic game Minesweeper. The creation of two different fields (display field and answer field) makes the players to ratiocinate the grids with mines and flag the hypothetical mines. Zcheck and Revealcheck enhance the game-play logic, these functions help players to avoid unnecessary operation. Furthermore, the different choices of difficulties and sizes make this game suitable for all stages of players.

An aesthetic weakness in the text-based version involves the way in which the field is displayed to the user. Since unclicked squares appear as 9s, the display field inherently has an awkward, cluttered look to it. Furthermore, the coordinate-based way of choosing the next square is also a little awkward, since it may leave the user to do a lot of counting rows and columns. When looking at the grid, the x axis is also more elongated than the vertical axis, further complicating the process of choosing the next guess square. Other minor bugs include occasional problems with game-play, like rare events where a victorious game outcome isn’t registered by the system. The biggest future improvement for the text-based version would be to increase the overall feel of the game by creating a cleaner, more Minesweeper-esque design and by improving the player input process, maybe by including row and column numbers on the sides of the grid.

There were balance issues when calculating the final score and aesthetic issues in the Leaderboard itself. The main strength of the score calculation is that it gives a healthy bias toward a player who decides to play and win on a harder mode and larger grid: one can’t simply choose Easy Mode on a 1x1 grid, solve it in under a second, and hope to appear on the Leaderboard. This bias might be a little too pronounced, especially as the grid size increases and the time factor begins to become irrelevant due to its logarithmic nature. For shorter times (under about two minutes) the time factor is pronounced and balanced, but plateaus due to the nature of the equation. Another hypothetical problem would be if the user solved the grid in exactly 1 second ($\log(1) = 0$), in which case Matlab would display his score as infinity. An aesthetic problem with the Leaderboard itself is the inability to display the score vector and the username vector side by side, making it awkward to look at.

Although lot of weaknesses are found in the text-based version of Minesweeper, the game can be processed decently. The coding strategies we had used in the Minesweeper included all the skills we had learned this semester. By creating this game, critical lessons were learned about problem solving, debugging skills, and using coding syntax to verbalize those problems. Future applications such as aesthetic improvement in the gameplay loop and the leaderboard could be implemented.

Tutorial:

A major strength of the entire project is the ability of the tutorial to respond to the specific situation the user finds himself in. As a whole, the tutorial provides the essential and detailed instructions for the user to feel comfortable with guessing and flagging. The tutorial is very well-adapted with the text-based version of the game, allowing the user to begin the game immediately after taking the tutorial. Another strength of the tutorial lies in its efficiency to a certain extent that it does not repeat instructions once displayed.

A fundamental weakness of the the tutorial is the fact that it uses only Zcheck and not Revealcheck to set the display field. As a consequence, if the user, for example incorrectly flag spot (2,1) and attempt to reveal spot (1,1), the display field is reset such that spot(2,1) shows 1 instead of 17 (a flag). In the real game with Revealcheck being called, attempt to reveal spot (1,1) leads to revealing the unflagged mine and the user loses, which might confuses the user since such case is not addressed in the tutorial. Another weakness is the inability to provide the user with hints on where to flag if no 0-spot is opened prior to their flagging move. Another big weakness of the tutorial is that it is not adapted for GUI-based version.

For future improvement, Revealcheck should be implemented in the code. Its respective checking value should also be added. This adjustment also requires a long list of move combinations that causes the execution of Revealcheck function. To address the hint-related problem, another long list of different scenarios that are obviously indicative of the correct placement of flag is also required. Although the tutorial is not adapted for the GUI-based version, it is sufficient in giving the user the gist of how Minesweeper game is logically played.

The GUI: The results of the work on the GUI are exceptional. For going in without any knowledge on how GUIs work in matlab, object based coding, or even something that became so fundamental as global variables, the fact that it works as well as it does it fantastic. Not to mention the fact that it does its core job of making the game much more enjoyable to play and easier to understand. For all intents and purposes, the GUI was a success.

The GUI for our version of Minesweeper is far from perfect. To start with the simplest potential upgrade, instead of a simple time function it would be nice to have the leaderboard tied in. This could, hypothetically, be done with little effort by adding an over text box for the user to enter their initials in during the victory screen, then using that input and the time in the Leaderboard function. However, time constraints prevent this from all coming together. Another quick upgrade would be a button on the BootUp GUI that would bring you back to the text based main menu.

The major GUI upgrades are much harder to propose. For example, you can’t flag in the current version. This is because buttons in matlab don’t have a right-click functionality. In order to flag, another button for such a process would need to be placed somewhere on the GUI, thus changing the functionality of clicking a button. This quickly turns into a cascade of having to rewrite large sections of the code, and with how touchy it already is, it wasn’t worth risking. Unfortunately this means the GUI is still a little cluttered. However, the biggest issue with the GUI is simply the lag. The cause of this lag is unknown but likely come from a number of sources. The first being the fact that the GUI uses a handful of global variables. While any other coding language would be fine with this, Matlab sucks when it comes to how they optimize them. I personally don’t know the technical details on why they are so bad, but nearly everywhere you look, people say to avoid them as they kill your memory when the program runs. Another source of lag could come from the fact that the GUI needs to restart every time it updates. As processing begins to slow down, doing something relatively intense like this just compounds the issue. Regardless, it would be recommended that this is looked into and something is done to try and decrease the lag.

References:

None/Matlab help and forums

Contributions: (separate page)

Lai: Developed a function (RevealCheck) that modifies the game, which enhanced the play experience of the game.

Mary: Testing section (this section is not directly related to this assignment, but definitely poses different ways to think about the same code). To develop this code, I had to break down different sections of the text based game into functions and run tests on individual concepts to test their output. Minor editing.

Matthew: Designed the 2 field system that is the basis of how the game is played (answerfield v. displayfield). Developed the recursive checking function (Zcheck) as well as the answerfield builder code. Pieced together an Alpha build of the game. Developed the GUI and all associated code. Wrote the corresponding parts of the paper.

Ted: Created the specific formula by which the final game score is calculated, handling the time element of finishing the game and relating it to the field size and difficulty mode chosen. Adjusted the different variables to create an equation that appropriately balances the difficulty chosen. Created the leaderboard function, which displays the initials of the user along with their final score.

Tran: Created the tutorial function that teaches the fundamental rules and logics of playing a text-based Minesweeper game, which enables a complete Minesweeper novice to play the game immediately.

Appendix:

(Codes format: TUTORIAL + MOTHER CODE + EVERYTHING IN IT + TESTING TABLE +GUI)

Tutorial:

```
function tutorial()
global delcellnum
delcellnum=0;
size = 5;
guidefor0 = 0;
guidefor1 = 0;
guidefor2 = 0;
forcedtyping = 0;
teachguess = 0;
teachflag = 0;
hintflag = 0;
forceunflag = 0;
teachunflag = 0;
numbermines = 4;
type = 0;
placex = 0;
placey = 0;
complete = 0;
flaggednum = 0;
Guess = 1;
guess = 1;
g = 1;
```

```
G = 1;
Flag = 0;
flag = 0;
f = 0;
F = 0;
% create an answer field:
answerfield=zeros(5,5);
answerfield(1,1) = 10;
answerfield(1,5) = 10;
answerfield(5,3) = 10;
answerfield(4,5) = 10;
answerfield(1,2) = 1;
answerfield(3,4) = 1;
answerfield(3,5) = 1;
answerfield(4,3) = 1;
answerfield(1,4) = 1;
answerfield(2,1) = 1;
answerfield(2,2) = 1;
answerfield(2,4) = 1;
answerfield(2,5) = 1;
answerfield(4,2) = 1;
answerfield(5,2) = 1;
answerfield(5,5) = 1;
answerfield(4,4) = 2;
answerfield(5,4) = 2;
% create a display field:
displayfield = zeros(5,5);
for i=1:5
    for j=1:5
        displayfield(i,j) = 9;
    end
end


%Tutorial:
disp('Welcome to tutorial');


while complete==0
        disp(displayfield);
        %Teach guess
        if teachguess == 0
        teachguess =1;
        disp('The nines denote unrevealed spots')
        disp('Each spot hides a value (0-8 or 10)')
        disp('For each move, you will be asked to either "Flag" or "Guess"')
        disp('To start the game, just type "Guess"');
            %Force user to guess
             while(type ~= 1)
                    type=input('Do you want to "Flag" or "Guess"? ');
             end
               disp('The grid is numbered in such a way that it is the first quadrant of a')
               disp('coordinate grid where the bottom left is (1,1) and top right is (size,size)')
               disp('For example, the location of the spot in the top left corner is (1,5)')
               disp('You are free to choose whatever spot you believe to not contain a bomb')
        else
               type=input('Do you want to "Flag" or "Guess"? ');
               if forceunflag == 1 && teachunflag == 0
                   %force to unflag

                   teachunflag = 1;
                   forceunflag = 0;
                   forcedtyping = 1;
                   while(type ~= 0)
                   type=input('Do you want to "Flag" or "Guess"? ');
                   end
                   placex=input('Where?(x) ');
                   while(placex <1 || placex >5)
                   placex=input('Where?(x) ');
                   end
                   placey=(5+1)-input('Where?(y) ');
```

```
                while(placey <1 || placey >5)
                placey=(5+1)-input('Where?(y) ');
                end
                while displayfield(:,placex) ~=7
                placex=input('Where?(x) ');
                end
                while displayfield(placey,placex) ~= 7
                    disp('Choose a flagged spot to unflag')
                    placex=input('Where?(x) ');
                    while(placex <1 || placex >5)
                    placex=input('Where?(x) ');
                    end
                    placey=(5+1)-input('Where?(y) ');
                    while(placey <1 || placey >5)
                    placey=(5+1)-input('Where?(y) ');
                    end
                    while(displayfield(:,placex) ~=7)
                    placex=input('Where?(x) ');
                    end
                end
            end
        end
        %Give hint if first flag
        if type == 0 && teachflag == 1
            if hintflag == 0
            %Give hint if user have opened 0-spot
            if guidefor0 == 1
                    hintflag = 1;
                    if displayfield(1,1) == 9
                    disp('HINT: Look at spot (2,4). All of its adjacents spots except for spot(1,5)')
                    disp('have been shown to not contain any bombs. Since spot (2,4) is a 1-spot, spot(1,5)')
                    disp('must be hiding a bomb. You should flag spot(1,5)!')
                    elseif displayfield(5,3) == 9
                    disp('HINT: Look at spot (2,1). All of its adjacents spots except for spot(3,1)')
                    disp('have been shown to not contain any bombs. Since spot (2,1) is a 1-spot, spot(3,1)')
                    disp('must be hiding a bomb. You should flag spot(3,1)!')
                    end
            elseif guidefor1 == 1 || guidefor2 == 1
                    disp('A flag is denoted by 17')

            end

            end
        end
        %Get input from keyboard
        if forcedtyping == 1
            forcedtyping = 0;
        else
            placex=input('Where?(x) ');
            while(placex <1 || placex >5)
                placex=input('Where?(x) ');
            end
            placey=(5+1)-input('Where?(y) ');
            while(placey <1 || placey >5)
                placey=(5+1)-input('Where?(y) ');
            end
        end


%Flag Code
if type==0

        if flaggednum<numbermines
        %Remove Flag
        if displayfield(placey,placex)==17
            displayfield(placey,placex)=9;
            flaggednum=flaggednum-1;
        %Error for flagging an opened spot
        elseif displayfield(placey,placex) ~= 9
            disp('You cannot flag an opened spot!');
        %Add Flag
        else
            displayfield(placey,placex)=17;
            flaggednum=flaggednum+1;
        end
        else
            if displayfield(placey,placex)==17
            displayfield(placey,placex)=9;
            flaggednum=flaggednum-1;
            else
               disp('You''ve placed all your flags!');
               if teachunflag == 0
               teachunflag = 1;
               disp('To unflag a spot, choose to flag the flagged spot')
               forceunflag = 1;
               end
            end
        end
%Guess a point (function: Check)
elseif type==1



            if displayfield(placey,placex) == 17
                     disp('This spot is flagged');
            elseif answerfield(placey,placex)==10
                     if guidefor0 == 0
                     guidefor0 = 1;
                     disp(displayfield)
                     disp('You triggered a bomb (denoted by 10)!')
                     disp('You would have lost immediately if this was an actual game')
                     disp('Now just pretend you have not opened this spot and choose a')
                     disp('different one instead')
                     else
                     disp('You triggered a bomb!')
                     disp('Choose a different spot')
                     end
                     displayfield(placey,placex) = 9;
            else % open an empty spot
               [displayfield]= Zcheck(displayfield, answerfield,placex,placey,5);
               if answerfield(placey,placex) == 0
                     guidefor0 = 1;
                     disp('You have opened a 0-spot, meaning the spots that are vertically,')
                     disp('horizontally and diagonally adjacent to it are empty, i.e contain no bomb')
                     disp('Opening a 0-spot will prompt the revelation of other adjacent 0-spots')
                     disp('and the bounded nonzero empty spots')


               elseif answerfield(placey,placex) == 1
                     if guidefor1 == 0
                     guidefor1 = 1;
                     disp('You have opened an 1-spot. There is one spot among the spots that are')
                     disp('vertically, horizontally and diagonally adjacent to it containing a bomb ')
                     end
                else
                     if guidefor2 == 0
                     guidefor2 =1;
                     disp('You have opened a 2-spot. There are two spots among the spots that are')
                     disp('vertically, horizontally and diagonally adjacent to it containing bombs ')
                     end
               end
            end
            %TeachFlag:
            if teachflag == 0
                teachflag = 1;
                disp('Once you have revealed a reasonable number of spots that help you deduce')
                disp('where the bomb is hidden, you should flag')
                disp('If you manage to flag all the bombs and reveal all the empty')
                disp('spots, you WIN')
                disp('To learn more about how flagging works, next time when asked, choose Flag')
        end

else
    disp('Invalid Entry, Please Flag or Check');
end
if displayfield==answerfield+[7 0 0 0 7; 0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 7; 0 0 7 0 0]
   complete=1;
end
end
ret=0;
while ret~=1
ret=input('Congradulations, you finished the tutorial! Enter 1 to return to the main menu ');
end
StartTheProject();
end
```

Text-based Game:

```
function MineSweeperBeta
%Core Varriables
%Difficulty translator
Easy=0;
easy=Easy;
e=Easy;
Medium=1;
medium=Medium;
m=Medium;
Hard=2;
hard=Hard;
h=Hard;
%Other Variables
global failed
failed=0;
Flag=0;
Guess=1;
flaggednum=0;
passed=0;
g=Guess;
f=Flag;
Y=1;
y=1;
N=0;
%User Inputs
size=input('What size (squared) do you want to play? (single numeric input)');
difficulty=input('What difficulty do you want? [Easy, Normal, or Hard]');
%Post input deffinitions
dispfield=zeros(size,size);
answerfield=zeros(size,size);
numbermines=ceil(size^(1+(difficulty/4)));%how many mines in game? Check back here for balance
global delcellnum
delcellnum=0;
%Fill the feilds
%dispfield
for i=1:size
   for j=1:size
      dispfield(i,j)=9; %* is representing the unclicked position
   end
end
%AnswerField
n=numbermines;
while n>0
   xpos=ceil((size)*rand());
   ypos=ceil((size)*rand());
   if answerfield(ypos,xpos)~=10 %10 is the representation of a mine
      answerfield(ypos,xpos)=10;
      n=n-1;
   end
end
for i=1:size
   for j=1:size
```

```
        for l=-1:1
            for m=-1:1
                if i+m>=1 && i+m<=size && j+l>=1 && j+l<=size && answerfield(j,i)~=10 && answerfield(j+l,i+m)==10
                    answerfield(j,i)=answerfield(j,i)+1;
                end
            end
        end
    end
end
%User Interaction/Begining of Game
tic;
while failed==0 && passed==0
    disp(dispfield);
    type=input('Do you want to "Flag" or "Guess"? ');
    placex=input('Where?(x) ');
    placey=(size+1)-input('Where?(y) ');
    %Flag Code
    if type==0
        if flaggednum<numbermines
            %Remove Flag
            if dispfield(placey,placex)==17
                dispfield(placey,placex)=9;
                flaggednum=flaggednum-1;
                %Add Flag
            else
                dispfield(placey,placex)=17;
                flaggednum=flaggednum+1;
            end
        else
            if dispfield(placey,placex)==17
                dispfield(placey,placex)=9;
                flaggednum=flaggednum-1;
            else
                disp('You''ve placed all your flags!');
            end
        end
        %Guess a point (function: Check)
    elseif type==1
        if answerfield(placey,placex)==10
            failed=1;
        elseif dispfield(placey,placex)==17
            disp('This spot is flagged');
        elseif dispfield(placey,placex)==9
            dispfield=Zcheck(dispfield, answerfield, placex, placey, size);
        else
            dispfield=RevealCheck(dispfield, answerfield, placex, placey, size);
        end
    else
        disp('Invalid Entry, Please Flag or Check');
    end
    if flaggednum==numbermines
        if flaggednum==((size^2)-delcellnum)
            passed=1;
        end
    end
end
t=toc;
if passed==1
    disp('You Win!');
    [s_vect, u_vect]=Leaderboard(t, size, difficulty);
    pa=input('Play again (Y/N)? ');
    if pa==1

        MineSweeperBeta()
    elseif pa==0

        StartTheProject()
    end
else
    disp('Game Over Loser!')
    pa=input('Play again (Y/N)? ');
    if pa==1

        MineSweeperBeta()
    elseif pa==0

        StartTheProject()
    end
end
```

Zcheck:

```
function [dispfield]=Zcheck(df, af, px, py, size)
dispfield=df;
answerfield=af;
global delcellnum
if px<=size && py<=size && px>=1 && py>=1
if answerfield(py,px)==0 && dispfield(py,px)==9
    dispfield(py,px)=answerfield(py,px);
    delcellnum=delcellnum+1;
    for i=-1:1
        for j=-1:1
            dispfield=Zcheck(dispfield, answerfield, px+i, py+j, size);
        end
    end
else
    dispfield(py,px)=answerfield(py,px);
    delcellnum=delcellnum+1;
end
end
```

RevealCheck:

```
function [displayfield] = RevealCheck(displayfield, af, x, y, size)
global failed
count = 0;

for i = (x-1):(x+1)
    for j = (y-1):(y+1)
        if i<=size && j<=size && i>=1 && j>=1
            if displayfield (j,i) == 7
                count = count + 1;
            end

        end
    end
end
if displayfield(y,x) == count
    for i = (x-1):(x+1)
        for j = (y-1):(y+1)
            if i<=size && j<=size && i>=1 && j>=1
                if displayfield(j,i)~=7&&af(j,i)~=10

                    %df(i,j) = af(i,j);
                    Size = size;
                    displayfield = Zcheck(displayfield,af,i,j,Size);

            elseif displayfield(j,i)==7 && af(j,i)~=10
                failed=1;
                end
            end
        end
    end

%displayfield = displayfield;
end
```

Leaderboard:

```
function [u_vect,s_vect] = Leaderboard(t,size,difficulty)

% u_vect = ["AAA";"AAA";"AAA";"AAA";"AAA";"AAA";"AAA";"AAA";"AAA";"AAA"];
% s_vect = zeros(10,1);

c = 0;
s_vect=(load('leaders', '-mat', 's_vect'));
s_vect=s_vect.s_vect;
u_vect=load('leaders', '-mat', 'u_vect');
u_vect=u_vect.u_vect;
difficulty = difficulty + 1; % Modify Matthew's difficulty variable to yours
username = input('Please enter your initials ','s'); % User gives username
score = ceil((size^2*difficulty*200)/log(t)); % Calcuplates score

for i1 = 1:10
    if score > s_vect(i1) && c == 0
        temp_s_vect = s_vect;
        temp_u_vect = u_vect;
        s_vect(i1) = score;
        u_vect(i1) = username;
        s_vect(i1+1:10) = temp_s_vect(i1:9);
        u_vect(i1+1:10) = temp_u_vect(i1:9);
        s_vect = maxk(s_vect,10);
        u_vect = u_vect(1:10);
        c = c + 1;
    end
end
board={u_vect, s_vect};
celldisp(board);
save('leaders','s_vect','u_vect','-mat');
```

GUI:

BootUp

```
function varargout = BootUp(varargin)
% BOOTUP MATLAB code for BootUp.fig
%      BOOTUP, by itself, creates a new BOOTUP or raises the existing
%      singleton*.
%
%      H = BOOTUP returns the handle to a new BOOTUP or the handle to
%      the existing singleton*.
%
%      BOOTUP('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in BOOTUP.M with the given input arguments.
%
%      BOOTUP('Property','Value',...) creates a new BOOTUP or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before BootUp_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to BootUp_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help BootUp

% Last Modified by GUIDE v2.5 21-Nov-2018 18:16:26

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @BootUp_OpeningFcn, ...
                   'gui_OutputFcn',  @BootUp_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before BootUp is made visible.
function BootUp_OpeningFcn(hObject, eventdata, handles, varargin)
movegui('center');
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to BootUp (see VARARGIN)

% Choose default command line output for BootUp
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes BootUp wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = BootUp_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
global Alive
Alive=1;
global s
s=get(handles.popupmenu1, 'Value')*2;
difficulty=get(handles.popupmenu2, 'Value')-2;
global ansfield
ansfield=ansfieldbuilder(s, difficulty);
global deleted
deleted=zeros(s);
%disp(ansfield)
global victory
victory=0;
GUI3;
tic;
%Gui4x4;
```

```
close(BootUp);
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1



% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2



% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
End
```

GUI3

```
function varargout = GUI3(varargin)
% GUI3 MATLAB code for GUI3.fig
%      GUI3, by itself, creates a new GUI3 or raises the existing
%      singleton*.
%
%      H = GUI3 returns the handle to a new GUI3 or the handle to
%      the existing singleton*.
%
%      GUI3('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI3.M with the given input arguments.
%
```

```
%      GUI3('Property','Value',...) creates a new GUI3 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI3_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUI3_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES


% Edit the above text to modify the response to help GUI3


% Last Modified by GUIDE v2.5 29-Nov-2018 12:26:46


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI3_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end


if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before GUI3 is made visible.
function GUI3_OpeningFcn(hObject, eventdata, handles, varargin)
movegui('center');
global s
global ansfield
global hands
global deleted
hands=handles;
num=0;
for i= 1:740/s:740
    for j=1:740/s:740
        num=num+1;
        posx=mod(num, s);
        if posx==0
            posx=s;
        end
        posy=ceil((num)/s);
        val=(ansfield(s-posx+1,posy));
        if val==0
            color=[0.5,0.5, 0.5];
            weight='normal';
        elseif val==1
              color=[0, 0, 1];
              weight='normal';
        elseif val==2
            color=[0.75, 0.3, 0.3];
            weight='normal';
        else
            color=[1,0,0];
            weight='bold';
        end
        val=num2str(val);
        txt = uicontrol('Style', 'text', 'BackgroundColor', [1,1,1], 'FontWeight', weight, 'ForegroundColor', color, 'String', val,...
            'Position', [i j 740/s 740/s]);
        if deleted(s-posx+1, posy)==0
            str1='posx=;';
            str2='posy=;global Alive; global deleted; h=gco;';
            str5='num=;buttondown(posx, posy, num); ';
            str3=insertAfter(str1, 5, num2str(posx));
            str4=insertAfter(str2,5, num2str(posy));
            str6=insertAfter(str5,4,num2str(num));
            str7=[str3 str4 str6];


            btn = uicontrol('Style', 'pushbutton',  'String', ' ',...
                'Position', [i j 740/s 740/s],...
                'Callback', str7);
            %'BackgroundColor', [0,0,0],
        end
    end
end

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI3 (see VARARGIN)

% Choose default command line output for GUI3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = GUI3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
```

EndGame

```
function varargout = EndGame(varargin)
% ENDGAME MATLAB code for EndGame.fig
%      ENDGAME, by itself, creates a new ENDGAME or raises the existing
%      singleton*.
%
%      H = ENDGAME returns the handle to a new ENDGAME or the handle to
%      the existing singleton*.
%
%      ENDGAME('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ENDGAME.M with the given input arguments.
%
%      ENDGAME('Property','Value',...) creates a new ENDGAME or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before EndGame_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to EndGame_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES


% Edit the above text to modify the response to help EndGame


% Last Modified by GUIDE v2.5 06-Dec-2018 16:43:03


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @EndGame_OpeningFcn, ...
                   'gui_OutputFcn',  @EndGame_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end


if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before EndGame is made visible.
function EndGame_OpeningFcn(hObject, eventdata, handles, varargin)
movegui('center');
global victory
global Alive
global time
if Alive==1 && victory==1
    matlabImage = imread('Victory.JPG');
    image(matlabImage)
    axis off
    axis image
    str='Time: ';
    str2=insertAfter(str, 6, num2str(time));
    txt = uicontrol('Style', 'text', 'String', 'Victory!',...
            'Position', [215 275 45 15]);
    txt = uicontrol('Style', 'text', 'String', str2,...
            'Position', [367.5 213 70 25]);
    btn = uicontrol('Style', 'pushbutton', 'String', 'Play Again?',...
                'Position', [367.5 300 70 25],...
                'Callback', 'BootUp;close(EndGame);');
else
    matlabImage = imread('Explosion.JPG');
    image(matlabImage)
    axis off
    axis image
    txt = uicontrol('Style', 'text', 'String', 'Ooof, Try Harder Next Time',...
            'Position', [360 330 150 15]);
    btn = uicontrol('Style', 'pushbutton', 'String', 'OK',...
                'Position', [400 300 70 25],...
                'Callback', 'BootUp;close(EndGame);');
end
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to EndGame (see VARARGIN)


% Choose default command line output for EndGame
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);


% UIWAIT makes EndGame wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = EndGame_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Get default command line output from handles structure
varargout{1} = handles.output;
```

CheckGUI

```
function checkGUI(posx, posy, num, handles)
global deleted;
global ansfield;
global s;
global Alive;
Alive=1;
% disp(posx)
% disp(posy)
if posx<=s && posy<=s && posx>=1 && posy>=1
    if deleted(s-posx+1, posy)==0
        if ansfield(s-posx+1,posy)==8
            Alive=0;
        elseif ansfield(s-posx+1,posy)==0
            %delete(hObject);
            deleted(s-posx+1, posy)=1;
            %           str1='pushbutton';
            %           str2='_Callback';
            %           p1=insertAfter(str1,10,num2str(num));
            %           %p2=insertAfter(str2,29,num2str(num));
            %           str3=[p1 str2];

            for i=-1:1
                for j=-1:1
                    nnum=num+i+j;

%                           posy=ceil((nnum)/s);
                        checkGUI(posx+i, posy+j, nnum, handles);
                        %                       str1='pushbutton';
                        %                       str2='_Callback';
                        %                       p1=insertAfter(str1,10,num2str(nnum));
                        %                       %p2=insertAfter(str2,29,num2str(num));
                        %                       str3=[p1 str2];

                    %end
                end
            end
        else
            deleted(s-posx+1, posy)=1;
        end
    end
    %delete(hObject);

end
% disp(deleted)
end
```

ButtonDown

```
function buttondown(posx, posy, num)
```

```
global Alive; global hands;  h=gcbo; global s; global deleted
global victory
global numbermines
global time
checkGUI(posx, posy, num, hands);
if sum(sum(deleted))==s^2-numbermines
    victory=1;
end
if Alive==1 && victory==0
GUI3;
elseif Alive==1 && victory==1
    close(GUI3);
    time=toc;
    EndGame

else
    close(GUI3);
    EndGame;
    toc;

end
```

StartTheProject:

```
function StartTheProject
disp(' ')
disp('Hello and Welcome to Minesweeper! Presented by:')
disp(' ');
disp('                Lai Wei');
disp('              Mary William');
disp('            Matthew Steigauf');
disp('              Ted Schmitz');
disp('                  and')
disp('                Tran Lu');
disp(' ');
disp(' ');
disp('Game Modes: (1)Turtorial, (2)Text Based, (3)Graphics Based, (4)Close');
disp('');
disp('What game mode do you want to play? Type the corresponding number and press Enter');
c=0;
while c==0
type=input('Game mode ');
if type==1
    c=1;
    tutorial();
elseif type==2
    c=1;
    MineSweeperBeta();
elseif type==3
    c=1;
    BootUp();
elseif type==4
    c=1;
else
    disp('Invlaid Input');
end
end
```

Testing (experimenting on how we could generate a testing code to see if the code does what we want it to do):

```
classdef MineSweeperAlphasTest < matlab.unittest.TestCase
    methods (Test) % testing dificulty calculations.
        function A(testcase)
            size = 4;
            difficulty = 1;
            realA = Difficultymines(size,difficulty);
            testedA = 6;
            testcase.verifyEqual(realA,testedA);
        end
        function B(testcase)% testing dispfeild calculation.
            size = 4;
            %difficulty = 1;
            realB = sum (FillWith9 (size));
            testedB = [9*size 9*size 9*size 9*size];
            testcase.verifyEqual (realB,testedB);
        end
        function C(testcase)% testing the number of mines Vs difficulty.
            size = 4;
            difficulty = 1;
            realC = Difficultymines(size,difficulty);
            C = Answerfeild (4,1);
            testedC = 0;
            for i= 1:4
                for j = 1:4
                    if  C(i,j)==10
                        testedC = testedC+1;
                    end
                end
            end
            testcase.verifyEqual (testedC,realC);
        end
        function D(testcase) %checks that a random location in the answerfeild has
            % the numbers of mines around it.
            size = 4;
            %difficulty = 1;
            D = Answerfeild (4,1);
            m = ceil(rand*size);
            j = ceil(rand*size);
           while D(m,j)==10 % && m < size && j < size && m > 0 && j > 0
             m = ceil(rand*size);
             j = ceil(rand*size);
           end
           realD = D(m,j);
           D1 = 0;
           for i = -1:1
              for l = -1:1
                  if  m+i<= size && j+l <= size && m+i >= 1 && j+l >= 1 && D(m+i,j+l)==10 && D(m,j)~=10
                  D1 = D1+1;
                  end
              end
           end
           testedD = D1;
           testcase.verifyEqual (testedD,realD);
        end
    end
End
```

Function(s) needed to run the tests:

1

```
function  [answerfield, xposv, yposv]= Answerfeild (size, difficulty)
            answerfield=zeros(size,size);
            n = Difficultymines(size,difficulty);
            xposv = zeros();
            yposv = zeros();
while n>0 % we are filling the matrix with mines calculated in line 20
    xpos=ceil((size)*rand());
    ypos=ceil((size)*rand());
    if answerfield(ypos,xpos)~=10 %10 is the representation of a mine
        answerfield(ypos,xpos)=10;
        n=n-1;
    end
end
for i=1:size % rows
```

```
            for j=1:size % columns
               for l=-1:1 % goes left right center -1 left 0 center 1 right [assume]
                  for m=-1:1 % goes up down center -1 down 0 center 1 up  [assume]
                     if i+m>=1 && i+m<=size && j+l>=1 && j+l<=size && answerfield(j,i)~=10 && answerfield(j+l,i+m)==10
                        answerfield(j,i)=answerfield(j,i)+1;
                     end
                  end
               end
            end
end
end
```

2

```
function numbermines = Difficultymines (size,difficulty)
numbermines=ceil(size^(1+(difficulty/4)));
end
```

3

```
function dispfield = FillWith9(size)
         dispfield=zeros(size,size);
         for i=1:size
            for j=1:size
               dispfield(i,j)=9;
            end
         end
      end
```