ENGR 0012 – Spring 2019
MATLAB Project

Acceptable behaviors for this assignment include:

- Asking your team members
- Asking your professor or TA
- Using MATLAB's built-in help and documentation
- Discussion with other teams/students is acceptable so long as no code is directly shared.

Unacceptable behaviors for this assignment include:

- Copying the solution(s) from a solution manual, book, other written material, or from other students
- Examining implemented code in MATLAB or any other language
- Providing the solutions to a classmate, student in other section, student in future section, or online solution banks

INTRODUCTION

In this project you will be creating a zero-player game called Game of Life. Game of Life, also known as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970. (source: Wikipedia.org)

"The game is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves, or, for advanced players, by creating patterns with particular properties." (source: Wikipedia.org)

The game is devised in a 2-D orthogonal grid of square cells, each of which is in one of two possible states: alive or dead. Each cell interacts with its eight neighbors (see Figure 1: Middle cell has 8 cells as neighbors.)

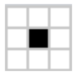


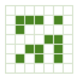
Figure 1: The black cell shown is surrounded by 8-neighborhood cells. Its state depends on the state of these 8 neighbors.

At each step in time, the cells undergo the following transition (all at once):

1. Any living cell with fewer than two live neighbors dies: underpopulation death
2. Any living cell with two or three live neighbors lives on to the next generation
3. Any living cell with more than three live neighbors dies: overpopulation death
4. Any dead cell with exactly three live neighbors becomes a live cell: reproduction.

The initial pattern is called the seed of the system. At each step the above rules to all cells simultaneously, and a new generation spawned based on its output.

Over several generations (iterations) the system may stabilize into a static pattern, periodic pattern, system death (collapse of the civilization).

Figure 2 shows an animation representing 50 generations of a simple seed pattern  where green = alive, white = dead

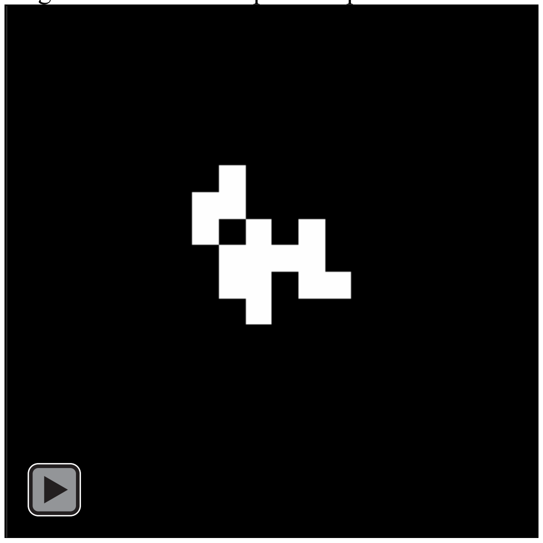


Figure 2: Animation of generations starting from Pattern described earlier.

INSTRUCTIONS
You will create several functions to accomplish the above task. The instructions in this sheet are practice exercises for you to begin thinking about the main project.

Practice Exercise 1:
Creating a gray scale image file in MATLAB
Recall that images are stored as numeric arrays. Gray Scale images are stored as 2-D numeric array containing rows and columns of data stored in data type of UINT8—this data type can only store a maximum value of 255 and a minimum value of 0.
Further, 0 represent black, and 255 represent white, with numbers between 0-255 representing darker to lighter shades of color.

Trace the code below:

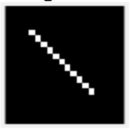
```
Code:

clear;
clc;
close all;

image = zeros(20,20);
for i = 5:15
    image(i,i) = 255;
end

imshow(image)
```

Output:



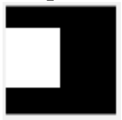
```
Code:

clear;
clc;
close all;

image = zeros(20,20);
for i = 5:15
    image(i,1:10) = 255;
end

imshow(image)
```

Output:



Try creating a larger image (100,100) with different patterns of white and black. Remember 255 represents white, 0 represents black.

Practice Exercise 2:
The illusion of movement using MATLAB figure windows.
Animation or any motion picture is in essence a rapid flickering of still images (~24 for realistic motion). By plotting the same figure in a figure window rapidly in succession, you can in effect generate an animation.

Trace the code below:

```
Code:

clear;
clc;
close all;

image = zeros(20,20);
for i = 1:20
    image(i,i) = 255;
    pause(0.2)
    imshow(image)
end
```

```
Output:

<animated Image>
```

Can you create other patterns or animations? What happens when you change the value in pause function to 0.5?

Do you understand how the following functions or tasks can be performed?

1. pause
2. imread
3. imresize → you may need this to increase size of image window
4. clear
5. clc
6. close
7. figure
8. f = figure() … figure(f)
9. zeros
10. imshow
11. loops using for and while
12. step through an array
13. nested loops
14. Images stored as matrices containing numbers 0…255

PROJECT EXPECTATIONS AND INSTRUCTIONS

- You will use the property of numeric arrays and image handling functionality to display organisms on screen.
- You will then use the knowledge you have gained executing array manipulations.
- You are required to create functions as described below to receive maximum possible grade.
- You can use any pattern (seed image) to begin your simulation including creating your own.
- Code format and comments are expected and receive a portion of the overall grade.

Terms used hereafter:

1. **Organism**: a single cell within the array containing 0 (dead) or 255 (alive).
2. **Health**: state of an organism: dead or alive
3. **World** or **canvas**: the entire array representing all elements of the current state of all organism. For example, a 20x20 world may contain up to 20x20 or 400 organisms.
4. **Organism** or **Organism index**: location of the cell within the world.

Overall concept: You will create an array (or receive an array containing a pattern). This array will be represented as an grayscale image file. Each array element i.e. pixel represents a single organism. The pixel intensity determines whether the cell is alive or dead. White pixel is alive (value = 255) and black pixel is dead (value =0). You can display this image using image functions in MATLAB. By scanning the neighbors of each pixel and determining its state based on the rules of the game, you can sweep through and update a new array that displays state of each organism in generation 2. Each iteration generates a new world array that contains updated states of each organism.

FUNCTION REQUIREMENTS:

Create the following required functions. You may create additional functions, however, the functions described below will be included in the grading rubric:

FUNCTION 1: **org_health**(argument1, argument2)
Input:
argument1: indices in an array of size 1x2
argument2: the current world represented by a numeric array of any size
Output:
Output1: returns a scalar value of 0 for dead, 255 for alive.

org_health function takes in the indices of the organism and the current world state. It then scans the neighborhood of the organism represented by the index. Remember to scan i-1:i+1 and j-1:j+1 of the neighborhood of organism at I,j.

Boundary conditions: we will treat all four edges of the world as infinite edges that wrap around. Let's take a 10x10 world. If an organism exists at location [0,3] its neighbors are at [-1,2], [-1,3], [-1,4] and 6 other neighbors around it. Since location -1 does not exist, this wraps around to the other side i.e. -1 becomes 10 (wrap from top to bottom of the array). Therefore the neighborhood changes to: [10,2],[10,3], and [10,4]. The other neighbors remain intact. Similarly if the organism is at 10,10. It's neighbors include: [11,9],[11,10],[11,10],[10,11] etc. Since row 11 and column 11 don't exist, they wrap around to the other side. I.e. the neighbors change to [1,9], [1,10], [1,10],[10,1] etc.

After checking the conditions:

1. Any living cell with fewer than two live neighbors dies: underpopulation death
2. Any living cell with two or three live neighbors lives on to the next generation
3. Any living cell with more than three live neighbors dies: overpopulation death
4. Any dead cell with exactly three live neighbors becomes a live cell: reproduction.

Return either 255 for alive or 0 for dead from this function.

FUNCTION 2: **display_world**(argument1, argument2, argument3)

Input:

Argument1 = the current world represented by a numeric array of any size

Argument2 = figure handle pointing to the figure window to plot the world

Argument3 = character vector representing the title to be displayed on the figure

Output:

NONE – however, the function displays information using graphical window.

display_world() function displays the current state on screen in a figure window set up for the purpose. This rapid display in same window creates the illusion of animation.

FUNCTION 3: **next_generation**(argument1)

Input:

Argument1 = the current world represented by a numeric array of any size

Output:

Output1 = updated world (called new_world)

Output2 = number of births

Output3 = number of deaths

next_generation() function steps through each element of the world array, invokes organism_health() to determine the status of an organism and uses this to build a new world of the same size as old but with updates organism life status. It also keeps track of the number of births and deaths that occur each time this function is called and returns these scalar values

Note: State change is applied simultaneously at each point. Check all the state changes in the world array and create a new world array with all the changes noted. This new world is generation two.

MAIN SCRIPT & OTHER FUNCTIONS:

You may create additional functions as needed, but you need to have functions 1-3 listed above to receive full credit.

In the main script:

1. Ask the user for file name pointing to the seed image (containing a pattern) to seed the Game of Life. This becomes the world array. Error check for file name and repeat indefinitely until correct file is selected.
2. Ask the user for number of iterations (generations). Error check the number to be above 1 and less than or equal to 2500
3. Set up a figure window to display seed world and all future iterations
4. Iterate through the number of generations while invoking necessary functions to update world array.
    a. Calculate total living population at each iteration and save it in an array that matches the number of iteration (same size as iterations)
    b. Store all births and deaths in corresponding arrays to track them at each generation
    c. Stop iteration if the state of world remains the same for current (n) and two generations prior (n-2)
5. Display the numeric statistics for the three variables: death, birth, population. These should include: mean and standard deviation, as well as median values. You may include other statistics, if find them appropriate.
    a. Graphically display information about death, birth, and population using a single figure window. You may choose to display three subplots or find an alternative method to display all these data. You should use at least one histogram and one plot command. These plots should have axis labels, title, and clearly display information.
6. Ask the user if they want to repeat this process, and if yes, close all windows, and start over from #1. Else display the following:
    a. Mean and standard deviations of population for each seed the user tried.
    b. File name of seed that created the largest population
    c. File name of seed that had the lowest population

Tips: Test your code to run with any seed image size. All seed images are grayscale and will result in a 2D array when imported.

REPORT REQUIREMENTS:
Finally, write a short report as team, reflecting on what you learned in developing this project. Include each team members' perspective on:

1. What did you find interesting, counterintuitive, or difficult in this project?
2. What did you enjoy the most? Why?
3. If given time and resources, how would you approach the problem differently? How would you redo it without the constraints of the functions requested?

SUBMISSION INSTRUCTIONS:
Include a statement on integrity in comments of your scripts and in footnotes of your documents. By submitting code, you are stating that:

1. The code was generated wholly by your team.
2. No part of the code was plagiarized, copied from a source, or discussed on a forum or online discussion board.
3. You did not help or receive help in writing code from anyone outside your team and instructors.
4. Peer discussion of the project did not include sharing code snippets.

We have set up tools to help us detect code similarity and plagiarism. Violations will be dealt with following First-Year Program and SSOE regulations and may result in Grade forfeiture, Academic probation, and up to Semester Suspension.

Name your script file: project1_teamNo.m
e.g. project1_teamL01.m

Name your function files appropriately.
e.g. org_health.m

Name your report as project1_report_teamNo.pdf
e.g. project1_report_teamL01.pdf

Compress all these files into a single .zip folder. Name this zip folder: InstructorName_Time_Project1_teamNo.zip
e.g. Mandala_10_Project1_TeamL01.zip → SUBMIT only the zip folder.

Submit your copies using Assignment Submission link found on desktop computers in classrooms. Selected Homework link under your class instructor link.