CMSC 201 Spring 2019

Project 3 – Minesweeper

Assignment: Project 3 – Minesweeper
Due Date:
Design Document: Friday, May 3rd, 2019 by 11:59:59 PM
Project: Friday, May 10th, 2019 by 11:59:59 PM
Value: 80 points

Collaboration: For Project 3, **collaboration is not allowed** – you must work individually. You may still come to office hours for help, but you may not work with any other CMSC 201 students.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
# File:    FILENAME.py
# Author:  YOUR NAME
# Date:    THE DATE
# Section: YOUR DISCUSSION SECTION NUMBER
# E-mail:  YOUR_EMAIL@umbc.edu
# Description:
#    DESCRIPTION OF WHAT THE PROGRAM DOES
```

For Project 3 you will have to turn in a "design document" in addition to the actual code. The design document is intended to help you practice deliberate construction of your program and how it will work, rather than coding as you go along, or starting without a plan.

Instructions

For this project, you will be creating a single program, but one that is bigger in size and complexity than any individual homework problem. This assignment will focus on manipulating lists, calling functions, and recursion. For this assignment, more than any other this semester, **planning ahead and designing your program will be very, *very* important**!

The design for Project 3 is entirely up to you – suggestions are provided within the project description, but you are not required to use them.

At the end, your Project 3 file must run without any errors.
It must also be called proj3.py (case sensitive).

Additional Instructions – Creating the proj3 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

You should create a directory in which to store your Project 3 files. We recommend calling it **`proj3`**, and creating it inside a newly-created directory called **`Projects`** inside the **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1.

Objective

Project 3 is designed to give you practice with two-dimensional lists, creating and calling functions, and recursion. You'll need to use practically everything you've learned so far, and will need to do some serious thinking about how all of the pieces you need to create should fit together.

Task

You will be programming up a simplified version of the Minesweeper puzzle game. The goal of the game is to determine the location of all of the board's "mines," using clues about the number of neighboring mines in each field. In our version, the game is won if all of the mines are correctly flagged (and no mines are incorrectly flagged).

You can read more about the game on the Wikipedia page, and you can play an online version of the game on this webpage. (Note that their criteria for winning is *different* from ours, and requires uncovering all of the non-mine fields.) You should also take a look at the sample outputs for examples of how the game is played, won, and lost.

Your program will need to:

- Read in and store a minesweeper board from a file
- Populate the board with the "clues" about the number of "mines" neighboring each tile
- Allow the user to either "reveal" or "flag" a specific field
- Print out a version of the board that only shows the user the fields they have already revealed and/or flagged
- Fill out "islands" of empty space when an empty field is revealed
- Check for win and lose conditions on each turn, and end the program if the game has been won or lost

Specification

Prior to this assignment, **you should be familiar with the entirety of the Coding Standards**, available on Blackboard under "Assignments" and linked on the course website at the top of the "Assignments" page.

You should be commenting your code, and using constants in your code (not magic numbers or strings).
Any numbers other than 0 or 1 are magic numbers!

You will **lose major points** if you do not follow the 201 coding standards.

If you have questions about commenting, whitespace, or any other coding standards, please come to office hours.

Additional Specifications

For this assignment, **you must use recursion to fill out the "island" of empty space when an empty field is revealed.** No other functions are required to be recursive. You also must create and call **at least eight individual functions**. All other design decisions are up to you.

For this assignment, you do need to worry about "input validation." You may assume that the user will enter the correct type of input (for example, an integer if one is asked for, a float if one is asked for), but the input may be negative, outside of the allowable range, or "bogus" (in the case of strings).

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

It is also acceptable if your program crashes when a filename is entered that either does not exist, or has the wrong formatting for the minesweeper board.

Details

The program starts by asking the user for the filename that contains the minesweeper board, and reads in the file. It then runs until the game is won or lost. At each turn, it should prompt the user for a row and column, and then for an action: "`r`" for reveal or "`f`" for flag.

Reading in and Creating the Board

When the board is read in from the file, it will only have three characters:

- An octothorpe "`#`" the borders of the board
- An asterisk "`*`" a field where a "mine" is located
- A space " " any field other than a border or "mine"

Your program will have to go through the board it has read in, and determine the "clue" for each field by counting the number of mines it touches. (This only needs to be done for fields that aren't border or "mine" fields.) That number clue will need to be stored or remembered somehow, but the specifics of that are left up to you.

When shown to the player, the board has different characters that represent different types of fields.

ex.	Character	Meaning	Explanation
`#`	Octothorpe	Border	Border of the game board
`.`	Period	"Unknown" field (unknown contents)	Field the user has not chosen to reveal or flag yet
	Space	"Island," a field with no mines touching it	Field that has been revealed, but does not touch any mines
`8`	Numbers (1, 2, 3, etc.)	"Clue" about the number of mines near that field	Field that has been revealed, and touches at least one mine
`F`	Capital F	Field that has been flagged	Field that may possibly contain a mine
`X`	Capital X	Detonated "mine"	Field that contains a mine, and was revealed

Displaying the Board

The board should be displayed to the user starting with only empty fields and borders, and details should only be shown as the user flags and reveals fields. For example, the board file on the left would display as the board shown on the right when the game starts:

board2.txt

```
#######
# *   #
#   * #
#     #
#  ** #
#    *#
#######
```

board as displayed to user

```
     1 2 3 4 5
   # # # # # # #
1  # . . . . . #
2  # . . . . . #
3  # . . . . . #
4  # . . . . . #
5  # . . . . . #
   # # # # # # #
```

HINT: Think very carefully about how to handle these two different boards! This is a crucial part of your design, so spend some serious time on it! (Hint, hint: using **two** different boards is one possibility.)

At each turn, the updated board should be printed to the user, and the number of "mines" left should be printed out.
(It *is* possible for the number of "mines" left to be negative, if the user has flagged more fields than there are "mines.")

We have provided a `prettyPrintBoard()` function for you, that will print the board with the row and column numbers, and with the board spaced out to look more like a square (as seen on the right above).

You can get the file from Dr. Gibson's pub folder using the command

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/pretty.txt .
```

There's also an optional "enhanced" version of the function, that will display the flags as red and black – but you must have a constant called **`FLAG`** that it can match against. You can get that version with the command

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/pretty2.txt .
```

Playing the Game

If the user chooses to reveal the field, one of the following things may occur:

- If the field is a "**mine**," it will "explode," and the game is over
- If the field is a **number clue**, the number will be revealed
- If the field is an "**island**", the game will *recursively* figure out the "edges" of the "island"
- If the field is a **flag**, the field will not be revealed
  - Print a message that that coordinate must be un-flagged first
- If the field was **already revealed** to be a number clue or an "island," nothing will happen
  - "Nothing" also means that no message will be printed

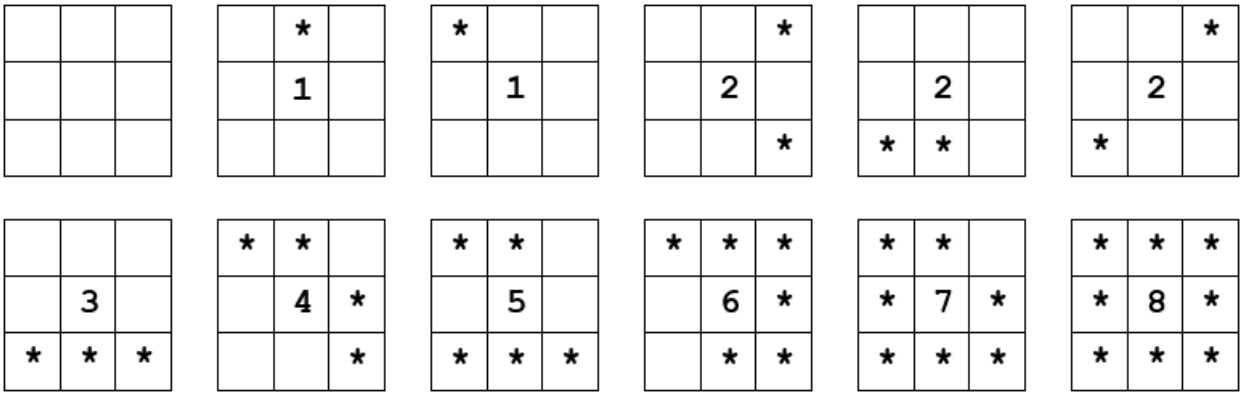
If the user chooses to flag the field, one of the following things may occur:

- If the field is **empty** (*e.g.*, unknown), a flag will be placed there
- If the field is **already flagged**, the flag will be removed
  - And the game will print a message stating that the flag was removed from that field
- If the field was **already revealed** to be a number clue or an "island," the field will not be flagged
  - And the game will print a message stating that that coordinate cannot be flagged

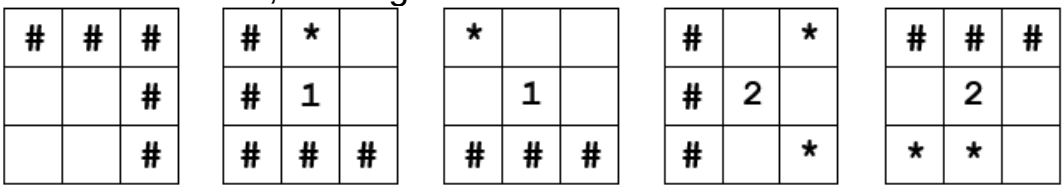
The game is won when all of the "mines" have been correctly flagged, and when none of the non-mine fields have been incorrectly flagged.

Number Clues

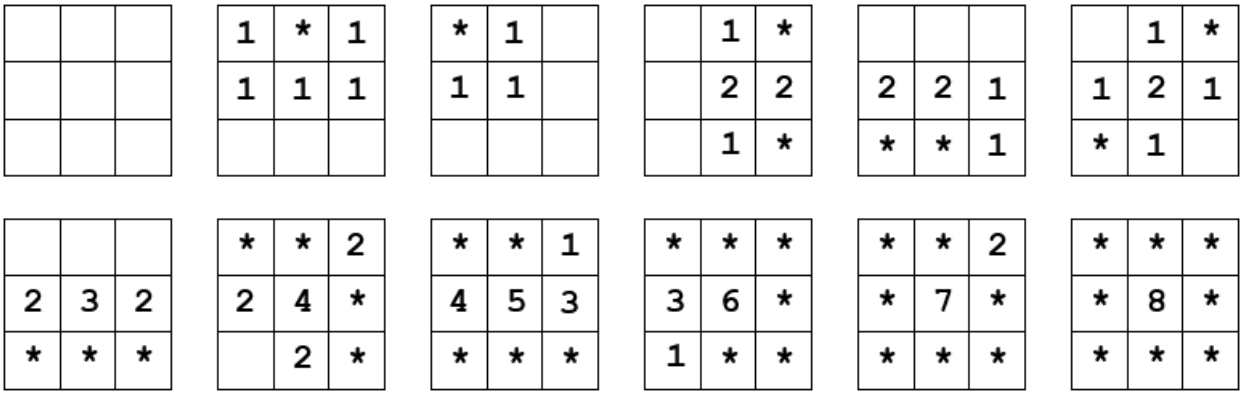
Fields that are not "mines" or borders will need to contain clues about how many "mines" are in the eight fields near them. Some examples are shown below, with the number clue in the center, and "mines" represented as a "*". (The top left has no "mines" nearby, so there is no number clue in the center.)



Note as well that the presence of border fields does not change the value of the number clue, as long as the same number of "mines" are still present.



The examples above only show the clue number for the center field. If these were complete boards, the clue list would look as follows (assuming there are no additional mines beyond the shown fields).



Revealing Islands

If the user chooses to reveal a field that is not a "mine" and has no "mines" near it (and has not already been flagged), then they have chosen to reveal an "island." When an "island" field is revealed, all of the neighboring islands and number clue fields also need to be revealed. For example, the game board state below was created only by revealing the field at row 4, column 1:

```
    1 2 3 4 5 6 7 8 9
  # # # # # # # # # #
1 # . . 1   1 . . . #
2 # 1 1 1   1 1 . . #
3 #           1 . . #
4 #           1 . . #
5 #           1 . . #
6 #       1 1 1 . . #
7 #       1 . . . . #
8 # 1 1 1 2 . . . . #
9 # . . . . . . . . #
  # # # # # # # # # #
```

The function to reveal all of the neighboring "island" and number clue fields **must be implemented using recursion**. As always, think about what the base case(s) should be, and what the recursive case(s) should be.

Additional Information and Examples

For more information, look at the sample output files available. They contain an example of the required input validation, how flags work, a game that is lost, a game that is won, and different board sizes.

You can download all of the boards used in the sample output (plus an additional, larger one) by using the following command:

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/board* .
```

You are also highly encouraged to make your own test boards – you can create and share these with your classmates if you want as well.

Points

The project is worth a total of 80 points. Of those points, 10 will be based on your design document (creation of it and following it), 10 will be based on following the coding standards, and the other 60 will be based on the functionality and completeness of your project.

Design Document

The design document will ensure that you begin seriously thinking about your project way early on. This will not only give you important experience doing design work, but will help you gauge the number of hours you'll need to set aside to be able to complete the project. **Your design document must be called design3.txt.**

For Project 3, you are creating the design entirely on your own.
You **may NOT work with another student** to "brainstorm" a solution or discuss any general approaches or requirements. If you need assistance with the design document, come to office hours.

Your design document must have four separate parts:

1. A file header, similar to those for your assignments
2. Constants
    a. A list of all the constants your program will need, including a short comment describing what each "group" of constants is for
3. Function headers
    a. A complete function header comment for each function your plan to create, including the description, inputs, and outputs
4. Pseudocode for main()
    a. A brief but descriptive breakdown of the steps your main() function will take to completely solve the problem; note function calls under the relevant comment (if applicable)

Your design can follow the same general format as the design for Project 1.

Your **`design3.txt`** file will be compared to the **`proj3.py`** file that you submit. Minor changes to the design are allowed. A minor change might be the addition of another function, or a small change to **`main()`**.

Major changes between the design and your project will lose you points. This would indicate that you didn't give sufficient thought to your design.
(If your submitted design doesn't work, it is generally better to lose the points on the design, and to have a functional program, rather than turning in a broken program that follows the design. The ultimate decision is up to you.)

To submit your design document, use

```
linux1[4]% submit cs201 PROJ3_DESIGN design3.txt
Submitting design3.txt...OK
linux1[5]%
```

Sample Output

As always, the sample output is available as a separate file under "Assignments" on Blackboard, and is called "sample3.txt".
There is a (very long) solved game for board3.txt provided in "sample3_long.txt" as well.
(Yours does not have to match the sample output exactly, but it should be similar. Don't forget that the **`prettyPrintBoard()`** function is available for you to use!)

Submitting

Once your **`proj3.py`** or **`design3.txt`** file is complete, it is time to turn it in with the **`submit`** command. (You may also turn the design or project in multiple times, as you reach new milestones or complete each piece. To do so, run **`submit`** as normal.)

To submit your design file (which is due Friday, May 3rd, 2019 by 11:59:59 PM), use the command:

```
linux1[4]% submit cs201 PROJ3_DESIGN design3.txt
Submitting design3.txt...OK
linux1[5]%
```

To submit your project file (which is due Friday, May 10th, 2019 by 11:59:59 PM), use the command:

```
linux1[4]% submit cs201 PROJ3 proj3.py
Submitting proj3.py...OK
linux1[5]%
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can check that your project and/or design was submitted by following the directions in Homework 0. Double-check that you submitted your files correctly, since **an empty file will result in a grade of zero for this assignment.**