APS 105 - Computer Fundamentals

Lab 6: Minesweeper

Winter 2020

This lab will give you experience writing software that makes use of arrays. Your solution will be marked by your TA during your scheduled lab period, and will also be submitted electronically by the end of your scheduled lab period.

Your TA will mark your solution based on its style, its use of arrays, and your verbal answers to a few questions.

For background information, check your lecture notes and online (do a search "arrays in c").

Minesweeper Game [10 points]

The following description of the Minesweeper game is based on: https://en.wikipedia.org/wiki/Minesweeper_(video_game)

Minesweeper is a single-player puzzle video game. The objective of the game is to clear a rectangular board containing hidden "mines" or bombs without detonating any of them, with help from clues about the number of neighboring mines in each field.

The game originates from the 1960s, and has been written for many computing platforms in use today. It has many variations and offshoots.

The player is initially presented with a grid of undifferentiated squares. Some randomly selected squares, unknown to the player, are designated to contain mines. Typically, the size of the grid and the number of mines are set in advance by the user.

The game is played by revealing squares of the grid by clicking or otherwise indicating each square. If a square containing a mine is revealed, the player loses the game. If no mine is revealed, a digit is instead displayed in the square, indicating how many adjacent squares contain mines. The player uses this information to deduce the contents of other squares, and may either safely reveal each square or mark the square as containing a mine.

An example of a game in progress is shown in Figure 1:

Your task will **not** involve any game playing but you will be required to generate the underlying grid for a game of minesweeper.

Your program should read three input values. The first two, **m** and **n** represent the dimensions. The game will be played on an **m** by **n** board. The third input, **p**, is a double value between 0 and 1 which represents the probability that a cell contains a mine.

Your program should first produce an **m** by **n** grid where 1 indicates the presence of a mine and

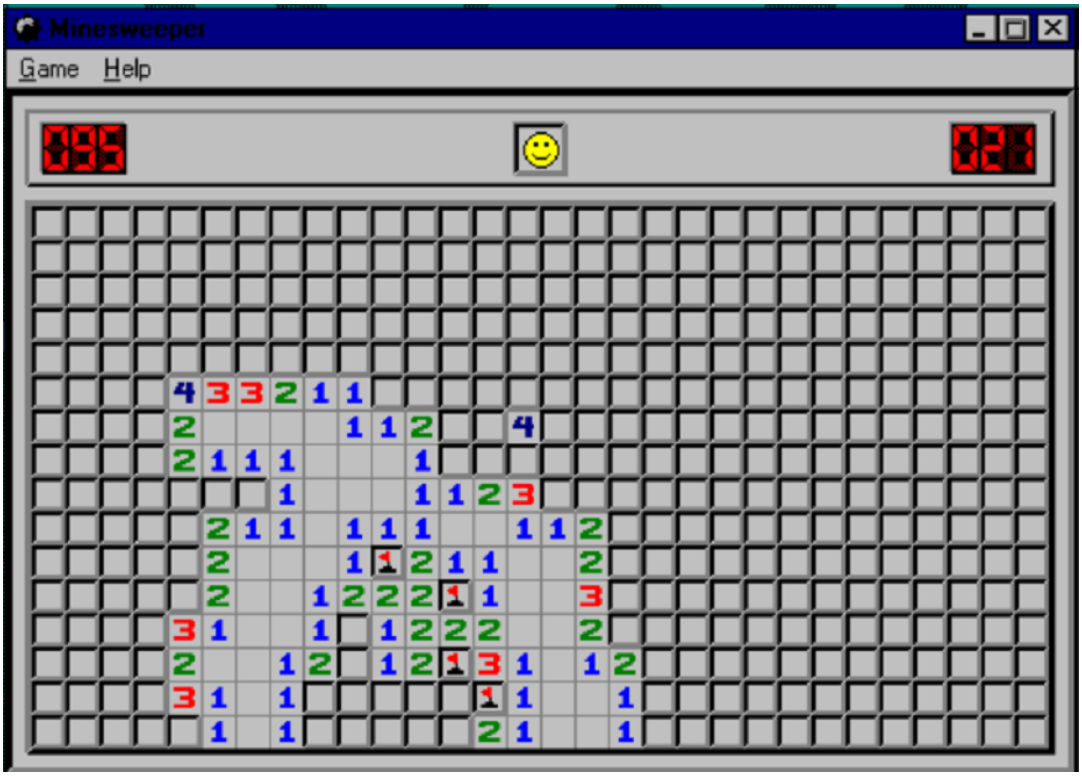

Figure 1: Trapezoidal method for integration

0 indicates a safe cell. For each cell, you should generate a random value between 0 and 1. If the value is less than **p**, you should place a mine in that cell. You should output this grid using an asterisk to indicate the presence of a bomb and a dot to indicate a safe cell. For example, if **m=5**, **n=10** and **p=0.3** the output could be as follows:

```
* . * . . . . . . .
* . . . . * * . * *
. . * * * . . . . .
* . . . * . . . . .
. . . . . * . * . *
```

Next generate and output the grid where each safe cell contains the number of neighboring bombs (above, below, left, right or diagonal). For the example above, the output would be:

```
* 3 * 1 1 2 2 2 2 2
* 4 3 4 3 * * 2 * *
2 3 * * * 4 2 2 2 2
* 2 2 4 * 3 2 1 2 1
1 1 0 1 2 * 2 * 2 *
```

Random number generator: For the sake of consistency of the output for the autograder, we

have provided a header file for the random number generator which is called "lab6.h". You should only include this file besides the `<stdio.h>` in the beginning of your program.

```
#include <stdio.h>
#include "lab6.h"
```

In your code, every time you need to generate a random number between 0 to 1, you may just call **rand()** function. For example,

```
double randomNum = rand(); //random number between 0 and 1
```

Please note that you should fill in the mine grid by calling the **rand()** function **row by row** to achieve the same grid as the test cases presented here and in the autograder. You should only call **rand()** function when it is needed.

Hint: To simplify the number of special cases, you should use an (m+2) by (n+2) array of integers. Rows 0 and m+1 as well as columns 0 and n+1 should be boundary cases which never contain bombs and are never printed but allow for an easier way of counting the number of neighboring bombs.

The Code to be Written:

1. You are to write a C program, in a file called `Lab6.c`

Sample Outputs

```
Enter the number of rows:  5
Enter the number of columns:  3
Enter the probability p value:  0.35

-----
* . *
. . .
. . .
. * .
. . *
-----
* 2 *
1 2 1
1 1 1
1 * 2
1 2 *
-----
```

```
Enter the number of rows:  6
Enter the number of columns:  10
Enter the probability p value:  0.67

------------------
* * * . * * * . . .
* . . * * * * * * *
. * * * * * * * * *
* * * * * * . . * *
. . . . * * * * . *
. . . . * * * * . *
------------------
* * * 4 * * * 4 3 2
* 6 6 * * * * * * *
4 * * * * * * * * *
* * * * * * 7 6 * *
2 3 3 5 * * * * 6 *
0 0 0 2 * * * * 4 *
------------------

Enter the number of rows:  7
Enter the number of columns:  12
Enter the probability p value:  0.2

----------------------
* . * . . . . . . * .
. . . * * . * * . . * *
* . . . . . * . . . * .
. . . . . . . . . * . .
. . . . . . . . . . . *
. * . * * . . . . . . .
. . . . . . . * . . . .
----------------------
* 2 * 3 2 2 2 2 1 2 * 3
2 3 2 * * 3 * * 1 3 * *
* 1 1 2 2 3 * 3 2 3 * 3
1 1 0 0 0 1 1 1 1 * 3 2
1 1 2 2 2 1 0 0 1 1 2 *
1 * 2 * * 1 1 1 1 0 1 1
1 1 2 2 2 1 1 * 1 0 0 0
----------------------
```

```
Enter the number of rows:  5
Enter the number of columns:  10
Enter the probability p value:  0.7

------------------
* * * . * * * . . .
* . . * * * * * * *
. * * * * * * * * *
* * * * * * . . * *
. . . . * * * * . *
------------------
* * * 4 * * * 4 3 2
* 6 6 * * * * * * *
4 * * * * * * * * *
* * * * * * 7 6 * *
2 3 3 4 * * * * 4 *
------------------

Enter the number of rows:  7
Enter the number of columns:  15
Enter the probability p value:  0.5

----------------------------
* . * . . * * . . . * . . . *
* * * * * . * * * * * . . . .
* . * * * . . . . * . . . . .
* * * . * . . . . . * * * . *
* * . * * . * * . . . . * * .
* * . . * . * . . . * * * * .
. . . * * . * * . . * . * . .
----------------------------
* 5 * 4 3 * * 4 3 4 * 2 0 1 *
* * * * * 5 * * * * * 2 0 1 1
* 8 * * * 4 2 3 4 * 5 4 2 2 1
* * * 7 * 4 2 2 2 2 * * * 4 *
* * 5 * * 5 * * 1 2 4 7 * * 3
* * 4 5 * 6 * 5 2 2 * * * * 2
2 2 2 * * 4 * * 1 2 * 5 * 3 1
----------------------------
```

Grading by TA and Submitting Your Program for Auto-Marking

The total of 10 marks on this lab are marked in two different ways:

1. **By your TA, for 4 marks out of 10.** Once you are ready, show your program to your TA so that we can mark your program for style, and to ask you a few questions to test your understanding of what is happening. Programs with good style have been described in previous labs. The TA will also ask you some questions to be sure that you understand the underlying concepts being exercised in this lab, and also check that you used a separate function for the cost calculation as described above.

2. **By an auto-marking program for 6 marks out of 10.** You must submit all of your program files through the ECF computers for marking. Long before you submit your program for marking, you should run the exercise program that compiles and runs your program and gives it sample inputs, and checks that the outputs are correct. Similar to previous labs you should run the following command:

   `/share/copy/aps105s/lab6/exercise`

   within the directory that contains your program.

   This program will look for the file **Lab6.c** in your directory, compile it, and run it on some of the test cases that will be used to mark your program automatically later. If there is anything wrong, the exercise program will report this to you, so read its output carefully, and fix the errors that it reports.

3. Once you have determined that your program is as correct as you can make it, then you must submit your program for auto-marking. This must be done by the end of your lab period as that is the due time. To do so, go into the directory containing your program and type the following command:

   `/share/copy/aps105s/lab6/submit`

   This command will re-run the exercise program to check that everything looks fine. If it finds a problem, it will ask you if you are sure that youwant to submit. Note that you may submit your work as many times as you want prior to the deadline; only the most recent submission is marked.

   The **exercise** program (and the **marker** program that you will run after the final deadline) will be looking for the exact letters as described in the output in this handout, including the capitalization.

   Important Note: You must submit your lab by the end of your assigned lab period. Late submissions will not be accepted, and you will receive a grade of zero.

You can also check to see if what you think you have submitted is actually there, for peace of mind, using the following command:

`/share/copy/aps105s/lab6/viewsubmitted`

This command will download into the directory you run it in, a copy of all of the files that have been submitted. If you already have files of thatsame name in your directory, these files will be renamed with a number added to the end of the filename.

After the Final Deadline — Obtaining Automark

Briefly after all lab sections have finished (after the end of the week), you will be able to run the automarker to determine the automarked fraction of your grade on the code you have submitted. To do so, run the following command:

`/share/copy/aps105s/lab6/marker`

This command will compile and run your code, and test it with all of the test cases used to determine the automark grade. You will be able to see those test cases output and what went right or wrong.