

# CS 1103-01 Programming 2 - AY2023-T1

[Dashboard](#) / [My courses](#) / [CS 1103-01 - AY2023-T1](#) / 8 September - 14 September / [Lab 4 Unit 2](#)

## Lab 4 Unit 2

### Lab 4: Recursion in MineSweeper

For this lab, you will be working with another application of recursion. You will add recursion to a nearly complete version of the well-known MineSweeper game.

For Minesweeper, you can copy the entire "mines" folder into your homework directory, even though you are only changing one of the files in that folder.

**You can start** the lab by creating a new project in your Eclipse workspace. Copy the folder **mines** from the source code directory into your Eclipse project.

Note that for this lab, the classes that you will be working with are defined in a package. The directory name "mines" is the same as the name of the package. This means that each class is defined in a file that begins with the declaration "package mines;", and the file is stored in a folder named mines. Since you know from the textbook (section 2.6.4, "The Problem of Packages") that the use of the default package is discouraged, it is a good idea to start getting used to creating packages. Fortunately, in Eclipse, it hardly makes a difference whether you use the default package or a named package. To compile or run the programs from the command line, it's a little more complicated (see the very last paragraph of section 2.6.4 in the textbook).

### Recursion in MineSweeper

In the usual (Microsoft) MineSweeper game, you click on squares that might or might not contain mines (only one mine per square). You know the total number of mines, but not which squares they are in. If you click on a mine, you get blown up. If you click on a square that is free of mines, you are told how many mines there are in neighboring squares. This number is shown inside the square, and if there are no mines in any of the neighboring squares then the square is just left blank (rather than showing a zero). "Neighbors" here are the vertically, horizontally, and diagonally neighboring squares, so a square can have up to eight neighbors. The color of the square changes when you visit it, so you can tell which squares have been visited. You win the game by visiting all the squares that don't contain mines, without visiting any mined square. To help you keep track of where the mines are, you can mark a square with a "flag" to show that you think it contains a mine. Clicks on a flagged square are ignored, so you can't accidentally blow yourself up by clicking one inadvertently. You can flag a square by right-clicking or shift-clicking it; you can remove a flag by right-clicking or shift-clicking it a second time.

There's another version of MineSweeper that works a little differently. In that version, you start at the upper left square of the minefield and have to make your way home, which is the lower right square, without getting blown up along the way. You can only visit a square that is next to one that you have already visited. ("Next to" here means above, below, to the left, or to the right; you can't move diagonally.) You win as soon as you safely reach the lower right square; you don't have to uncover every un-mined square. The lower right square is guaranteed to be free of mines. Also, to get you started, the upper left square and

its neighbors are also free of mines.

This is the version that you will work on for this lab.

The *mines* package already defines a working version of MineSweeper, but it is unsatisfactory in one respect: If you visit a square and the number of bombs in neighboring squares is zero, then you know automatically that it's safe to click on every neighbor of that square. Since it can be done automatically, without thought, it would be nice if the computer would just do it. Furthermore, if one of the neighboring squares also has no mined neighbors, then you can automatically visit all the neighbors of that square. And so on.... This is a recursive procedure, and your job is to add a recursive method to the program to implement it. (There are ways to do the same thing non-recursively, but they are not as efficient or nice.) A completed, compiled version of the program can be found in the jar file *MineSweeperComplete.jar* in the source code directory, so you can see how much nicer the game is after the recursion is added. **You should try the completed game before working on the program, to make sure that you know what is supposed to happen.**

Note that when you play the game, you start with a visited square in the upper left, and you have to work from there. Flagged squares, which you make by shift-clicking or right-clicking, are shown in pink with a "B" to indicate that you think there's a bomb there. If you step on a bomb, you lose the game, and then the locations of all the bombs are revealed by coloring them red.

The main method for MineSweeper is in the file *MineSweeper.java*, and you should **run that file** to see how it works. The game is actually implemented in the file *MineField.java*, and that is where you will be working. (The other file, *MineMenus.java*, implements the menu bar for the program, and you can ignore it.)

**For the major part of the lab, you only need to modify the method `mark(row,col)`, which is defined at the very bottom of *MineField.java*.** This method is called from elsewhere in the program when the user clicks on a square; when the method is called, it has already been checked that the user is allowed to click there and that there is no mine in that square. Currently, the `mark` method contains the single line `"state[row][col] = STATE_VISITED;"` which is meant to record the fact that the user has visited the square in the specified row and column. You have to modify `mark()` so that in addition to marking just the one square where the user clicked, it will **also** recursively mark neighboring squares as visited if it is known that they don't contain any mines. (You only have to mark the squares by setting a value in an array; you don't have to do any re-drawing, as that is already taken care of elsewhere.)

The *MineField* class uses the two-dimensional array named `state` to keep track of the state of each square on the board. The value of `state[r][c]` is the state of the square in row `r` and column `c`. The possible values are: `STATE_FLAGGED`, meaning that the user has flagged the square as probably containing a bomb; `STATE_VISITED`, meaning that the square has been visited and the user can see how many mines are in neighboring squares; and `STATE_UNVISITED`, which is the initial state and means that the square has not yet been visited or flagged.

There is also a two-dimensional boolean array named `mined` that records the positions of the mines. The value of `mined[r][c]` is true if there is a mine in the square in row `r` and column `c`. Note that a method `bombCount(row,col)` is already defined for counting the number of bombs in the neighbors of a given square, so you won't have to do this counting yourself. Remember that the basic idea is that if you visit a square at `(row,col)` and if `bombCount(row,col)` is zero, then you should also automatically visit all the neighbors of that square.

The recursive `mark()` method will be similar (but **not** identical!) to the "blob-counting" method that can be found in section 9.1.4 in the textbook. It will be rather short – you can do it in fifteen lines -- but it's tricky to get everything right.

Remember that you need base cases and that you have to avoid infinite recursion.

Last modified: Wednesday, 28 November 2018, 6:43 AM

---

## UoPeople Clock (GMT-5)

All activities close on Wednesdays at 11:55 PM, except for Learning Journals/Portfolios which close on Thursdays at 11:55 PM **always following the clock at the top of the page.**

Due dates/times *displayed* in activities will vary with your chosen time zone, however you are still bound to the **11:55 PM GMT-5** deadline.

---

### ◀ Lab 3 Unit 2

Jump to...

Code Unit 2 ▶


---

[Disclaimer Regarding Use of Course Material](#) - [Terms of Use](#)

University of the People is a 501(c)(3) not for profit organization. Contributions are tax deductible to the extent permitted by law.

Copyright © University of the People 2021. All rights reserved.

You are logged in as [Abel Lifaefi Mbula](#) ([Log out](#))

 [www.uopeople.edu](https://www.uopeople.edu)



### Resources

[UoPeople Library](#)

[Orientation Videos](#)

[LRC](#)

[Syllabus Repository](#)

[Career Resource Center](#)

[Honors Lists](#)

### Links

[About Us](#)

[Policies](#)

[University Catalog](#)

[Support](#)

[Student Portal](#)

### Instructors

[Instructor Portal](#)

[CTE Center for Teaching Excellence](#)

[Office 365](#)

[Tipalti](#)

### Contact us

[English \(en\)](#)

[English \(en\)](#)

[العربية \(ar\)](#)

[Data retention summary](#)

[Get the mobile app](#)