Refactoring Hints for 2D Mines

You are not logged in.

If you are a current student, please Log In for full access to the web site.
Note that this link will take you to an external site (`https://shimmer.mit.edu`) to authenticate, and then you will be redirected back to this page.

This document is intended to provide some hints for places to look as you are reorganizing the preexisting code for the 2D minesweeper game.

Of course, if you are still having trouble after using this document, there are plenty of opportunities to get help from a human (tutorials, lab sessions, office hours, the forum, etc.)!

You can also take a look at this document, which talks about some general aspects of style that we think are important.

Refactoring

> *"A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away."*
> -Antoine de Saint-Exupéry

1. In general, we will try to follow the so-called **DRY** principle (**D**on't **R**epeat **Y**ourself). Basically, the suggestion is that multiple fragments of code should not describe redundant logic. Rather, that logic can often be refactored to make use of a variable, a loop, or a function to avoid re-writing the same code multiple times.
2. If you find yourself re-writing the same short expression over and over, that might be a good sign that you can store that in a variable as an intermediate result. If you find yourself copy/pasting a block of code to compute a result, that might be an opportunity to define a function and/or to use a looping structure.
3. Also be on the lookout for redundant code. Code that, for example, performs multiple checks for the same condition, or reassigns a variable to have the exact same value, or something of that nature, can often be removed without affecting the behavior of the program.
4. Nested conditionals can often be rewritten as a single conditional to save on indentation by using `and` and/or `or`.

Hints for Some (but not all) Opportunities for Improvement

Below are some specific suggestions for areas in which the implementation in `lab.py` could be improved. You do not need to follow these guidelines, but if you are stuck looking for opportunities for improvement, you may find the following suggestions helpful. Note also that this does not contain suggestions related to *all* of the things that could be improved; just a few to get you started.

1. The code in `new_game_2d` for computing the number of neighboring bombs is massive, and the same block is repeated multiple times with little modification between repetitions. Try to restructure that piece of the code to avoid this kind of repetition.
2. In `new_game_2d`, the blocks of code related to creating `game['visible']` and creating `game['board']` are very similar: they each create a 2-d array of a particular size and fill it with appropriate values. Try to restructure this by making a generic "helper" function, and creating both the board and the "visible" array using that function.
3. Several blocks throughout the code attempt to find the neighbors of a particular cell (specifically, only those that are "in bounds") to perform some operation on them. Try to replace these blocks by defining a helper function to perform this operation, and replacing these blocks with calls to that function.

4. There are several blocks throughout the code that loop over the entire game structure to determine the new *state* of the game after an update. Try to restructure this into a single helper function, which can be used to test for the state of a game based on the board and "visible" array.