**CS 100  Programming I  
Project 3: Organization**

**John C. Lusth**

**Revision Date: October 2, 2015**

**Preamble**

You may develop your code anywhere, but you must ensure it runs correctly under a Linux distribution before submission.

**Program Organization**

You must follow and implement the functions and programs described in this document to receive any credit.

The stepwise refinement methodology of the previous programming assignment exhibited a *top-down* approach. In a top-down approach, the main function is sequentially refined until the project is finished. For this project, you should take a *bottom-up* approach, in which you start out by writing the most basic functions and then write higher-level functions that call these basic functions, and so on.

**Level 0**

A board can be represented as a two-dimensional array or matrix. A location on the board can be represented as a row and a column. Define a set of functions and procedures for allocating and populating a board, given the specifications from the command line.

* One function should allocate an empty board. Two boards will be allocated with this function, one which will hold mine information (the *mine* board) and one which holds the covered/uncovered status of each square (the *status* board).
* A second procedure should place the desired number of mines on a given *mine* board.
* A third function, when given a *mine* board and a location, calculates the number of mines adjacent to that location. It does this by checking each of its eight (or fewer) neighbors for mines.
* A fourth procedure should place the number of adjacent mines in each square in a given *mine* board (via the third function).
* A fifth procedure should initialize the given *status* board so that every square is considered covered.

For the *mine* board, you will need to decide on a value to represent a mine that is distinguishable from the count of neighboring mines. For example, you can't use the number 1 to represent a mine, since it could also represent a count. For the *status* board, you will need a value that represents a covered square and a value to represent an uncovered square.

Place these functions and procedures in a module named *game.py*.

Write a program named *level0.py* that thoroughly tests the board allocation functions and initialization procedures. This program should import the *game.py* module.

**Level 1**

Define a procedure that displays a board. This function is given two boards, a *mine* board and a *status* board. As the board is being drawn, the function checks the *status* board to see if a covering mark should be displayed or if the number of adjacent mines (found on the *mine* board) should be displayed.

Place this procedure in *io.py*.

Write a program named *level1.py* that thoroughly tests your display procedure. Your program should print the initial board, manually uncover a few squares, print the board, manually uncover all the squares, and print the board one final time. This program will need to import the *game.py* and *io.py* modules.

**Level 2**

Define an pyrotechnic display procedure. At the start, this procedure should just print the word BOOM.

This procedure should be placed in *io.py*.

Write a program named *level2.py* that prints the initial board, manually uncover a few squares, prints the board again, and calls the pyrotechnic display procedure. This program will need to import both *game.py* and *io.py*.

**Level 3**

Define an "uncovering" function that, when given a *mine* board, a *status* board, and a location, uncovers the square on the given board. If the square has no adjacent mines, the function recursively calls itself to uncover all the neighbors of the square. If the function uncovers a square with a mine, it should return 1. Otherwise, it should return 0.

Place this function in *game.py*.

Write a program named *level3.py* that prints the initial board, calls the uncovering function to uncover a square that should have a cascading effect, and prints the board again.

**Level 4**

Define a "end of game" checking function that, when given a *mine* board and a *status* board, returns 0 if all covered squares contain mines and 1 otherwise.

Write a program named *level4.py* that tests the end-of-game checking function thoroughly.

**Level 5**

Define a "move" function that, when given a *mine* board and a *status* board, immediately calls the end-of-game checking function. If it returns 0, then the this function should return 2. Otherwise, this function prompts for a location and then uncovers that square using the function from Level 3. If the uncovering function returns a 0, this function should display the board using the procedure from Level 1 and return a 0. If the uncovering function returns a 1, then the pyrotechnic display procedure from Level 2 should be called and a value of 1 returned.

Note that a return value of zero means the game should continue. A return value of 1 or 2 means the game is over.

Place this function in *io.py*.

Write a program named *level5.py* that tests the three return values of the move function.

**Level 6**

Define a "game loop" procedure that implements a classic "reading" loop for running the game. The initial *mine* and *status* boards should be passed to this procedure. Before the loop starts, the move function from Level 5 should be called and the return value saved. As long as this saved return value is 0, the loop should run. The body of the loop reruns the move function, resaving the return value. When the loop exits, either call the pyrotechnic display procedure or a congratulatory message.

Place this procedure in *minesweeper.py*, along with a *main* function. The *main* function should allocate the game boards and call the game-loop procedure.

Your final program should consist of *minesweeper.py*, *game.py*, and *io.py*.

**Level 7**

Refine the procedures from Level 1 and Level 2 to make a more spectacular game. Making system calls with the *clear* program can make the game look nicer.

Note, for each function you write at each level, you should use top-down stepwise refinement. That is to say, start with the simplest version of the function and successively add capability to the function. Save each program from each level.

Note also: you should see the following modules when you submit: *minsweeper.py*, *game.py*, *io.py*, *level0.py*, *level1.py*, *level2.py*, *level3.py*, *level4.py*, and *level5.py*.

The *level* functions will be tested and graded as well as your minesweeper program!

*lusth@cs.ua.edu*