Programming assignment A

ASCII Minesweeper

Official release on Feb 14th at 1pm (Document may change before then without notice)
Due 5pm Feb 25th

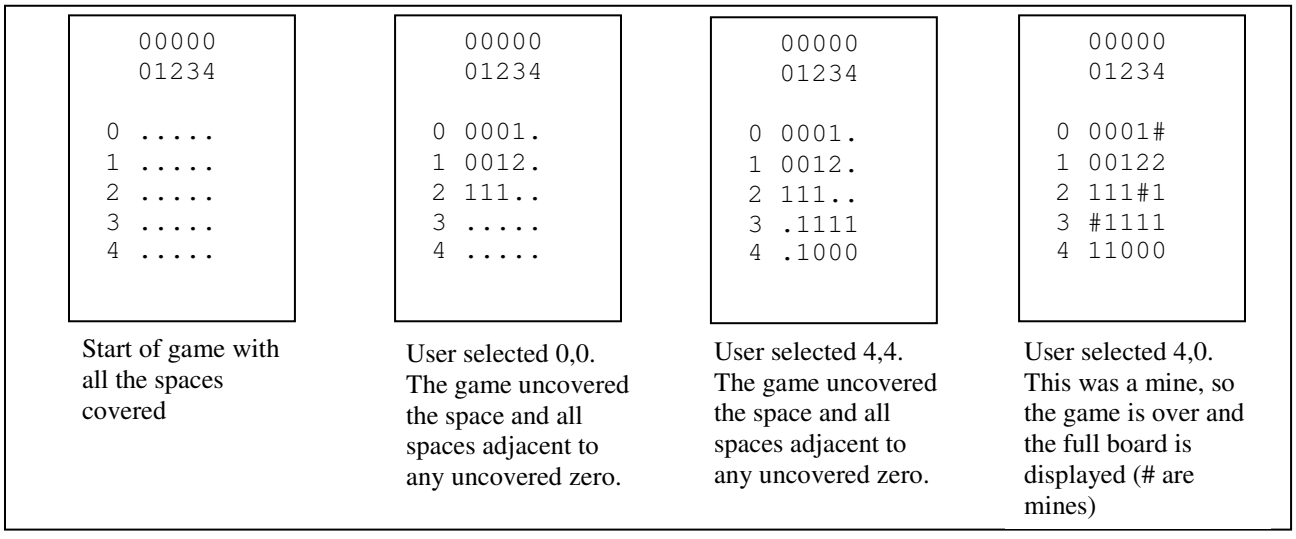
Minesweeper is computer game that was first written in 1989 by Robert Donner. The game became very popular when it was distributed with Windows 3.1. It still comes with all Windows computers. This project is worth approximately 6% of your class grade.

Game description

Note, this is a slightly modified version of a minesweeper description found on wikipedia http://en.wikipedia.org/wiki/Minesweeper_%28computer_game%29. You might find it helpful to go to that page to learn more about minesweeper.

> The game screen consists of a rectangular field of squares. Each square can be cleared, or uncovered, by selecting it. If a square that contained a mine is selected, the game is over. If the square did not contain a mine, two things can happen. A number will appear indicating the amount of adjacent (including diagonally-adjacent) squares containing mines. In addition, if there is no mine adjacent to that space, then the game automatically clears those squares adjacent to the empty square (since they could not contain mines). Further, anytime a space is adjacent (including diagonals) to an uncovered space with 0 mines adjacent to it, that space is also uncovered. The game is won when all squares that do not contain a mine are cleared.

In our version of the game the board is displayed as a text output. Here is a 5x5 board at different stages of completion.



Code provided

We have provided you with a main program and a few functions to give you a start on writing this program. The entirety of the code provided is available on the web. The main follows:

```
main(int argc, char * argv[])
{
    string fname="";  // Initialization not needed, but makes me happy.
    int map[COLS][ROWS];
    int found[COLS][ROWS];
    int done=0; // 0=not done, 1=player lost, 2=player won
    Move playerMove;
    int moveCount=0;
    int mines;


    if(argc==2)
        fname=argv[1];

    mines=generateMap(map,fname);  // if file="", generate random map.
                                   // Else generate map defined by file.
                                   // return the number of mines
    clearFound(found);
    while(!done)
    {
        displayMap(map,found);
        playerMove=getMove();
        done=updateMap(map, found, playerMove,mines);
        moveCount++;

    }
    printScore(done, moveCount, map);
}
```

Note that COLS and ROWS are previously declared as global constants. The program uses the following data structures:

- **map[][]** is a two dimensional array which specifies where the mines are as well as information about the number of adjacent mines
- **found[][]** is a two-dimensional array that is used in parallel with map[][]. If found[x][y]=1, this means that the location has been uncovered by the user and should be displayed. If it is equal to zero that means the user hasn't yet uncovered the space and when the map is displayed this space should be shown as a blank.
- **playerMove** is a struct which simply has the row and column the user has selected to uncover.
- **mines** is the number of mines in the map.

The following functions are either written for you, partially written, or you need to write them from scratch.

- **generateMap()** will either generate a map from a file (if the user specifies one from the command line) or will generate a random map. If it is a random map you should prompt the user for the number of mines he or she wishes to have. Assuming that number is possible, you map must have that many mines! **generateMap()** will also set the non-mine spaces equal to the number of mines adjacent to the space. If there is a mine in the space it will set the space's value equal to MINE (which is a global constant set to be 100). The part where it reads from a file written for you. In all cases, the number of mines on the map will be sent as the return value.

- **clearFound()** sets the entire found array to be zero (so no spaces start uncovered). You need to write this function.

- **displayMap()** displays the uncovered parts of the map. We provide this function but you really should read it carefully so that you understand what it does and what it is expecting. You shouldn't change this function unless you need to for some optional feature.

- **getMove()** prompts the user for the row and column they wish to uncover. It needs to check to be sure that the space actually exists on the map (so the row and column numbers aren't too big or too small. getMove() should tell the user if they enter an invalid row or column and ask them to try again. *Optional: you can give the user the option of quiting.* If the user hits a mine (or quits) you should return a 1. If the user has uncovered all non-mine spaces, the function should return a 2. (They've won).

- **updateMap()** figures out what spaces to uncover (and changes the found array as needed) based on what the user asked to uncover. This may include more than one space if the uncovered space was a zero. You need to write this function.

- **printScore()** should at the least print the entire map (uncovered) and tell the user if he or she won or lost. If you want to compute a score based upon the number of moves or something else, you are welcome to do so.

Hints, help and clues

Perhaps the most important thing is **Start early**. By Monday the 21st you should have the vast majority of this done and just be debugging. Debugging will take much longer than it has in the past.

You really only have the following tasks:

- Complete generateMap(). Note, as written this involves a call to a function called findMineValues() which would populate the non-mine spaces with the number of adjacent mines(0 to 8)
- Write clearFound() (this is pretty easy)
- Write getMove(). While this is harder than clearFound(), it is still not too bad.
- Write updateMap(). (This is the hardest part by far)
- Write printScore (also quite easy, although you need to display the whole uncovered map, which might be a bit of a pain).

As noted, updateMap() is the hard part, with generateMap() being a bit tricky. Once you understand the code we've given to you, generateMap() is really the only other difficult function. *You really should work on the easy parts first.*

Hints and clues:

- This project can be quite intimidating. The trick is to think of it (and of most programming) as a group of functions. We've already done the high-level design for you—you "just" need to write the functions!
    - Read the code first. While the code isn't complete, you should be able to use the skills you've gained in reading code to understand what is going on. Take your time and read things carefully.
    - Start with the easy ones. clearFound(), getMove() and printScore() are all pretty simple once you understand the big picture.
- For generateMap(), you will need to generate random numbers. You saw how to generate random numbers back when we used random numbers find an estimate of Pi. You don't need to worry about seeding the random number generator.

    - Also, you need to be really careful when counting the adjacent mines that you don't go outside of the map array bounds. This can be tricky.
- For updateMap() it is strongly suggested you start by just uncovering the location specified by the user and returning a 1 if they have hit a mine. Once you have that (and all the rest of the program working) you can work on uncovering everything adjacent to an uncovered 0. Getting the "uncover all spaces next to an uncovered zero" feature working is quite difficult.
    - Once you get everything else working, you need to think about how to uncover all spaces next to an uncovered zero. The hard part is being sure that you have uncovered all the squares you should. There are a number of ways to do this, but perhaps the easiest is:
        - If the player's move uncovered a zero, walk every square on the board. If that square is next to an uncovered zero, uncover it. Once you have done that with the whole board figure out if you uncovered any zeros during that walk. If you did, you will have to walk every square again. You continue until one walk of all of the squares resulted in no new squares being uncovered.
- Use the –Wall flag when compiling. Really.
- Be very careful with your array bounds. Always be checking (with asserts or whatever else) that you aren't going to go past the array bounds. As a general rule, look over every access to an array and convince yourself that there is now way the array can go past its bounds. Debugging problems like this are very hard indeed. (When Prof. Brehob wrote this assignment, he spent the better part of an hour looking for array-bound related errors. And he has a lot of experience finding these things. If you have an error like this you can easily get stuck until you get help.) In general if something really wacky is going on, you probably have an array bound error. Even if the error appears unrelated to an array in any way.

Rules for the project

Your code is to be based upon the code we have provided. You are welcome to make changes to our program. The only really key thing is that you must be able to read the exact same input files the code we gave you can read. We would greatly prefer you use our displayMap() function, but if you wish to add a feature we don't support (see optional parts, below) then (and only then) you are allowed to make changes to displayMap().

As always this code is to be written on your own. We will be running an autocorrelator to look for those who have copied code from each other. You may not copy code from any source other than our texts. See the course web page for further clarification on collaboration. If in doubt, ask the instructor.

You will hand in the project by placing your code in a directory called "PA". The ONLY .cc files that should be in that directory is your code (it can be in multiple files if you wish). When in your PA directory, the command:

```
g++ *.cc
```

should generate a file named "a.out" which is your minesweeper game.

Optional features

- User has a way of marking a mine. If they are wrong they lose, otherwise the mine is uncovered. This would probably by having the user specify that they are uncovering something (Say the user types "U 0 0" to uncover 0 0. They lose if that space is a mine. Or the user might type ("M 0 0") to indicate they believe that 0 0 is a mine. They lose if it isn't a mine.)
- User has a way of saving the game and reloading it latter. Probably involves writing a file which describes the board as well as a file specifying which spaces have been uncovered. There must be a way to restart the game after saved, even if the program has been quit.
- Seed the random number generator. (look around for information on srand() and time().)
- Something else cool.

The first 2 items would likely be worth 10 points each depending on how well you implemented it. The 3rd might be worth around 3 points. You can't get better than 110 points on the project. Further, small extra things (like allowing the user to quit early) will not be worth any points, although they may make the game better (and easier to debug). *If you have any optional features you must create a file named "optional.txt" which describes these features. This file must be in your PA directory.*

Scoring:

- Program works, but you don't uncover all spaces next to an uncovered 0. **80 points**
- Program works entirely. **100 points**
- Program works and you did some optional things. **Up to 110 points**

You can lose points (up to 20) for poorly written (or poorly commented) code. Further, if you turn it in after its due date/time, but before Tuesday March 8th at 5pm you will lose 20 points. After that, you will get zero points.