Minesweeper

CS32 Assignment 1

Due Dates:

Assignment Out:	Jan 22, 1998
Design Due:	Jan 27, 1998 (11:59pm)
Minesweeper Due:	Feb 5, 1998 (11:59pm)

1.0 Introduction

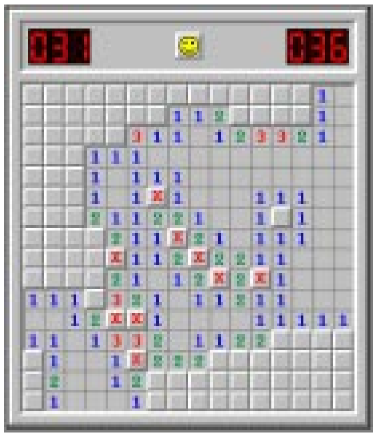
Over summer, Andy Miller and Sam Hazlehurst decided that they would realize their childhood dream of founding a game company. Their plan was to choose a niche so small that they couldn't fail to dominate it: Minesweeper games. When Yours & Mine Software finally shipped MinesweeperPro in September, they met with disappointing sales. Apparently, a dominant software vendor was shipping a similar product as part of its operating system.

Y&M SW's budget legal staff tried to argue in Hazlehurst vs. Microsoft Corp., 1997, that MS's practice of packaging Minesweeper with its OS was anti-competitive and monopolistic, but Microsoft, represented pro bono by Allen Ginsberg, convinced 7 Supreme Court Justices that "Minesweeper is an integral, inseparable part of the Windows NT Operating System." In this assignment, you will learn to commiserate with Sam and Andy by writing your own implementation of Minesweeper. Or you fail.

We hope that Minesweeper will be a worthwhile introduction to C++ for those new to the language, and a useful refresher for those who have used it in the past. Additionally, we hope to rejuvenate your design skills, which have probably laid dormant for the last month, and to acquaint you somewhat with the UNIX development environment.

2.0 The Game

Minesweeper is easiest to learn by playing. For a good, if somewhat unstable, implementation, check out `xmine`. Minesweeper consists of a 2-dimensional grid of squares. Each square is originally covered; under some squares there are mines. When you **uncover** a square with a mine under it, you lose. When you **uncover** a square that doesn't have a mine under it, the game displays the number of mined squares adjacent to the current square (a number from 0 to 8). If this number is zero, the game must perform the **uncover** operation on all squares adjacent to it.



A user **flags** a square to indicate that she thinks that there is a mine underneath the square. When all the squares on the board which are not mines have been uncovered, the player has won the game.

A "power-feature" of Minesweeper that many are unaware of is the ability to **check** a square. When the user checks an uncovered square, and there are as many flagged squares adjacent to the checked square as there are mines (i.e., the player believes she has found all the mines adjacent to this checked square), the game performs the **uncover** operation on all the unflagged squares adjacent to the square. If there are more or less flagged squares adjacent to the checked square than mines, or if the player checks a square that is covered, nothing happens. This all sounds pretty complicated, but playing with it in `xmine` or the demo will clarify what we mean. In order to check a square in `xmine`, middle-click on it.

Minesweeper supports varying dimensions for the board; the larger the board, the more difficult the game. The game has an 8 x 8 beginner level; a 16 x 16 intermediate level; and a 16 x 30 advanced level. The game also keeps track of the time that a player takes to complete a game, and uses these times to maintain three high-score lists (one for each level). High score lists consist of the ten highest scores and the players who achieved them.

Your Game

You must implement your Minesweeper with a text-only interface, a la `/course/cs032/demos/minesweeper/mine-text`. Your minesweeper must allow the user to uncover, flag, and check squares. Your game must implement all three levels of difficulty, and allow the user to choose which level to play. You must time players' games, and keep three separate high score files, one for each difficulty level. These high score files can be in any format you care to use, as long as your program can both read and write that format. Your high score list should contain at least ten players.

The high score list will obviously require that you use some kind of sort algorithm. You should be able to write a sort routine if you've taken the prerequisites to CS 32.

You must provide some textual way for the user to represent their commands. The demo might give you some ideas for this. You're free to use any input format for commands that makes sense. Be sure to document your choices well enough in your README for us to test your code.

You have the option of linking against a TA support library to implement a breath-taking 3D Minesweeper, but if you decide to do so, **you still must hand in a working text-only version**. If you are interested in doing the graphical Minesweeper, consult the documentation for the support code, since your design may be significantly impacted by this decision.

Manual Pages to Read

CS 32 expects a greater measure of self-sufficiency than any CS course you are likely to have taken up to this point. Specifically, we will often be referring you to more thorough documentation, rather than providing complete documentation ourselves. This is one of those times. The on-line manual, available via the `man` command (and on the web via the Answerbook --- see the CS32 Documentation web page), will be your best friend this semester. For future reference the notation `foo(`*`n`*`)` means "the manual page for `foo` in section *n* of the manual." You can see this page by typing `man -s `*`n`*` foo`. Here are some `man` pages you may find helpful in writing Minesweeper:

- **`time(2)`** will let you get the time in seconds since January 1, 1970. You will need this, or something similar, to time the player's games.
- **`rand(3c)`** is a random number generator. You will need some sort of random number generator to populate your board with mines. **`drand48(3c)`** also works well.

Your books also have documentation on topics such as text and file I/O, so make sure you use them as references as well.

Building Your Game

You'll be using Sun's WorkShop Pro 4.2 C++ compiler, `CC`, to compile all of your C++ programs in CS 32. For this assignment, you will be invoking `CC` directly from the command line.

Compiling a single file program with `CC` is easy enough. You will need to specify the `-g` flag, which directs `CC` to prepare your code for the debugger, and the `-o <filename>` flag, which causes `CC` to emit an executable named `<filename>`. So, to compile a program whose source is in `Snork.C` into an executable named `snork`, you could use the following command:

```
cslab3e ~/src -> CC -o snork -g Snork.C
```

Then typing `./snork` will run your program. Multi-file programs aren't too difficult either:

```
cslab3e ~/src -> CC -o snork -g NormalSnork.C SuperSnork.C
```

Design Check

We will be expecting you to turn in a design check for this program. Minesweeper does not have an overly complex design, but you should get into the habit of designing the entirety of a program before you start coding it. We expect from you a neatly written paper handin of the design for your program, comprised of a class diagram and list of important methods of classes. Additionally, a description of the flow of control of the program is required. You are not required to use OMT or Rational Rose for this handin. The important thing is that it is neat and readable.

Also, keep in mind that if the design takes you one day to do, you don't have to (and should not) wait until after the design handin date to start coding. If you wish to talk to a TA about your design before you start writing code, come see one during TA hours. Especially if you are new to C++, you are better off starting early and having time to deal with any problems you encounter, rather than having trouble in the 11th hour before it is due.

Handing In

You must hand in an electronic version of your assignment as well as a hardcopy version. Type `cs032_handin minesweeper` in the directory that contains all your source and your `README` file to hand in your electronic version. You should print out all your source code and your `README` using the `cis` printer and hand it in within 24 hours after the deadline for the electronic handin.

Some Things to Remember

It's always a good idea to start early, but especially if this is your first C++ program, be sure to give yourself enough time. "Enough time" will vary widely from person to person, so why take the chance?