Test Program 2

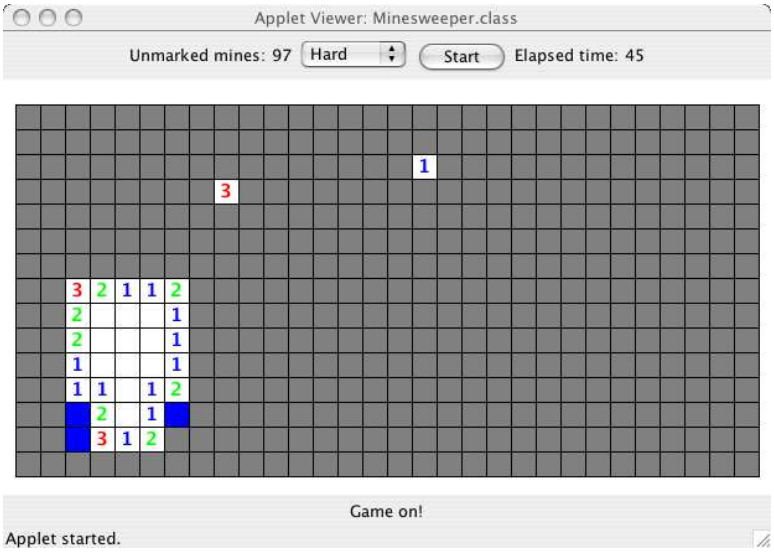
Handout 13
CSCI 134: Spring 2005
April 27

Design Due: Wednesday, May 4, start of class
Due: Friday, May 13, 4 PM.

Minesweeper

Minesweeper (along with its partner, Solitaire) has been responsible for countless hours of procrastination and lost productivity since it was included in Microsoft's Windows 3.1. The game seems simple enough. The board is divided into a grid and "mines" are hidden under some grid cells. The player's task is to determine where all the mines are without causing them to explode. If the player clicks on a grid cell and it contains a mine, the mine explodes, ending the game. To make the game remotely winnable, when a grid cell is clicked on and there is no mine, the cell is labeled with the number of neighboring cells that do have a mine. From this, the player can often learn more about the location of mines, and place a flag on the locations where mines are believed to exist.

Below is what the grid might look like during the game:



The main part of the window displays the grid. As the image indicates, a grid cell may be:

- Exposed. An exposed cell either contains a number or is empty (that is, a white rectangle). The number indicates how many of the cell's 8 neighbors contain mines. (A neighbor is the cell immediately to the left, right, above, below, or any of the four cells touching a corner of the cell. Cells on the outer edge of the grid have fewer than 8 neighbors.) An empty rectangle means that none of the cell's neighbors contain mines.
- Marked. A marked cell is depicted as a blue rectangle. The player marks a cell when he/she believes he/she knows where there is a mine.
- Neither marked nor exposed. These are depicted as gray rectangles. The player has no information about the contents of these cells yet.

In addition to the information shown to the user, a cell may also:

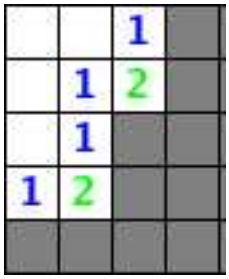
- Contain a mine or not. A mine could be in a marked cell or an unexposed cell, but not an exposed cell, because the act of exposing a cell would detonate the mine and the player would lose at that point.

Game Strategy

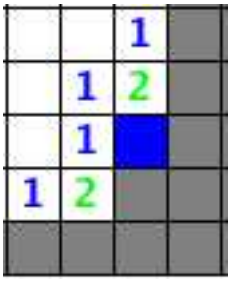
You do not need to understand the strategy outlined in this section to be able to implement the game, but a little understanding will make playing it more fun.

In the image on the first page, the player has clicked on a few cells randomly to expose them. (You can't do anything more intelligent than that at the start of the game.) After exposing some cells, you may find a cell with no mines in neighboring cells. At that point, you know it is safe to click on each of its neighbors. In fact, the game saves you this tedium and exposes them all for you. So, when you get lucky and click on a cell with no neighbors an area will open up that is empty in the middle and outlined with numbers around the edge. You can now look for patterns in these numbers to give you clues about where mines are.

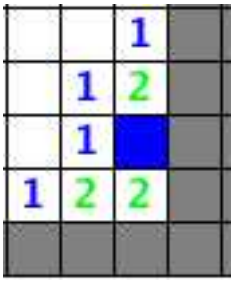
In particular, corners can give us very useful information. For example, consider the corner that looks like this near the bottom left of the image on the first screen:



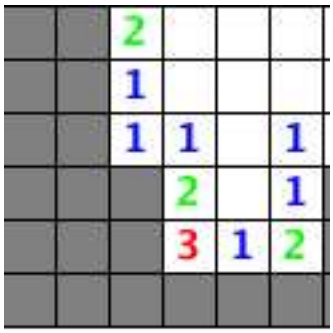
The "1" in the second column and second row indicates that the cell has one neighbor with a mine. In this case, we know the contents of all its neighbors except for one. We can thus conclude that the unexposed cell at its lower left corner contains a mine. This would be a good cell for the player to mark:



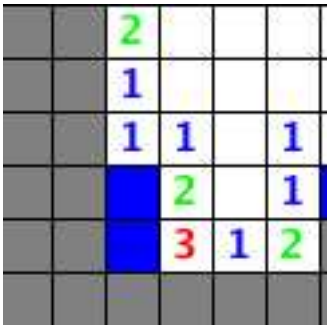
Now, we can consider the "1" in the third row and second column. We know it has exactly one neighboring mine, and we have marked one of its neighbors to indicate a mine. Therefore, we can expose the rest of its neighbors, in this case just the cell directly below the one we marked:



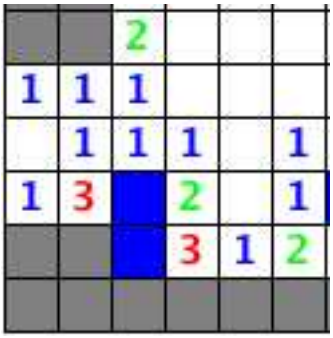
Now, we move a little to the left and examine this configuration:



The “1” in the third row and fourth column indicates that the cell has 1 neighbor with a mine. As above, we can mark its only unknown neighbor (at its bottom-left corner). But here, we know even more. The “2” in the fourth row and fourth column has only two neighbors unexposed. Both of these must contain mines, so the player can mark both of these:



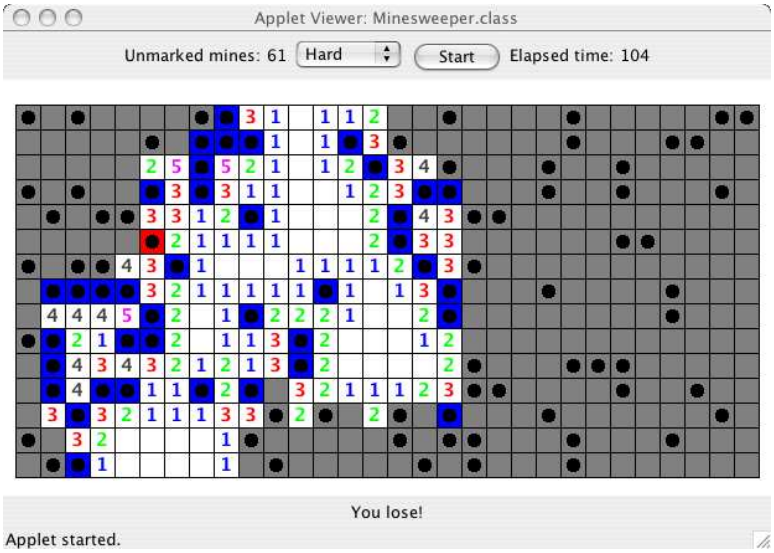
Again, we have learned some useful information. The cell marked with a “1” in the third row and second column has one neighbor marked, so we know that there are no mines in its other neighbors. We can expose its three unexposed neighbors. This exposes another “1” in the third row, second column. We also know where its one neighboring mine is, so we can expose its other neighbors as well:



These are only some of the simplest patterns you can learn to look for.

Displaying mines

Below is an image showing what the game might look like when the player loses:



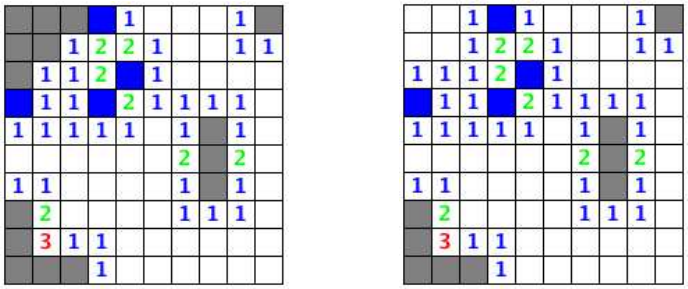
In this screen shot, in addition to the information mentioned earlier, the locations of the mines are also shown by drawing black circles in the cells that contain mines. Notice that some of the cells containing mines were marked, while some were unexposed. The user apparently attempted to expose a cell that turned out to contain a mine. If you look at the online version of this document, you will see that one of the mined squares (at row 6, column 6) has a red background. This is the location where the user clicked incorrectly and exposed a mine. (It has a slightly different shade of gray in the print version.)

Playing the Game

When the game starts, all the cells are unexposed. The user can play this game with a 2-button mouse. Using a 1-button mouse, the user will either click or hold the control key down while clicking. The game proceeds by the user clicking on cells, where mouse clicks have the following effect.

Mouse click (left button on 2 button mouse)

- If the cell contains a mine, the player loses the game.
- If the cell contains a mark (but no mine), nothing happens.
- If the cell is unexposed (and does not contain a mine), it is exposed, revealing the number of its neighbors that contain mines. If none of its neighbors contain mines, all of its neighbors are exposed (recursively).

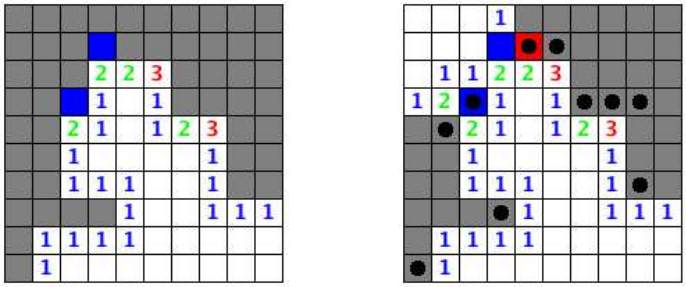


For example, consider the game situation pictured on the left above. Here, the player can safely expose any of several cells in the upper left corner of the board. Suppose the player clicks on the cell in the first row, second column. Note that the player can safely do this because the “1”

in row 2, column 3 has its one neighboring mine marked. Since this newly-exposed cell has no neighboring mines, all of its neighbors are also exposed. So in the first row, the cells in the first and third columns are exposed and in the second row, the cells in the first and second columns are exposed. But there is more to do. Since the now-exposed cell in the second row, first column has no neighboring mines, its neighbors are also exposed automatically. Here, the only one of its neighbors that has not already been exposed is the "1" in row 3, column 1. This "automatic recursive exposure" can continue for several levels if there are many cells clustered together which don't have mines.

- If the cell is exposed and the number on the cell equals the number of its neighbors that are marked, all of its neighbors are exposed. The previous example also demonstrates this behavior. If the player clicks on the "1" in the third row, third column, the game should notice that this is a cell with one neighboring mine and one marked neighbor. So all of its unexposed neighbors will be exposed. In this case, that is just the cell at row 2, column 2. And after that 0-neighbor cell is exposed, the automatic recursive exposure described above takes over and exposes the entire corner.

  A misplaced mark, however, can result in an attempt to expose a cell that contains a mine. This will end the game. Consider the situation on the left in this example, where the player has mistakenly marked the cell in the second row, fourth column.



  Then, noticing that the "2" in the third row, fourth column has two marked neighbors, the player clicks on that "2" cell. The game will the expose all neighbors of that cell, but unfortunately, one of those cells, the one in row 2, column 5, has a mine, which has been exposed and colored red. The game ends in a loss.

Control-Mouse click (right button on 2 button mouse)

- If the cell is unexposed, it is marked.
- If the cell is marked, the mark is removed.
- If the cell is exposed, nothing happens.

None of the control-mouse click behaviors depends on whether the cell contains a mine.

The Assignment

For Test Program 2, you will write a version of Minesweeper, as described above. You can find a link to our working implementation at

`http://www.cs.williams.edu/~cs134/handouts.html`

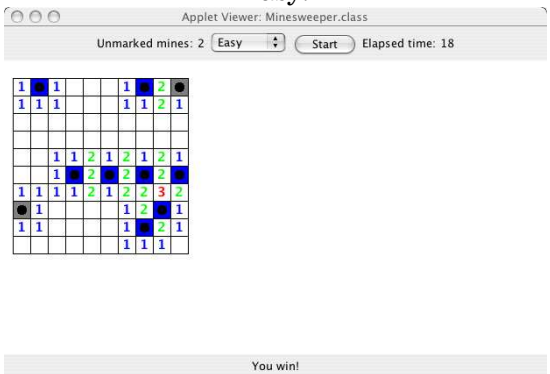
Your version of the game will include the same basic functionality as the Windows version. In addition to the handling of mouse input, your game should have the following functionality:

- A game begins when the start button is pressed. The grid is created and the mines are randomly placed on the board. The contents of all grid cells are initially unexposed.

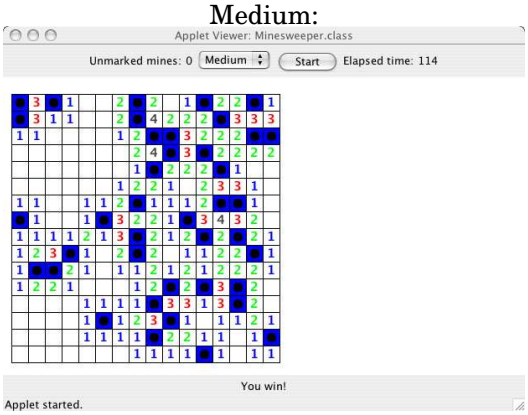
- The size of the grid and the number of mines is determined by a difficulty selection made using a `JComboBox`.

Level	Rows	Columns	Mines
Easy	10	10	10
Medium	16	16	40
Hard	15	30	100

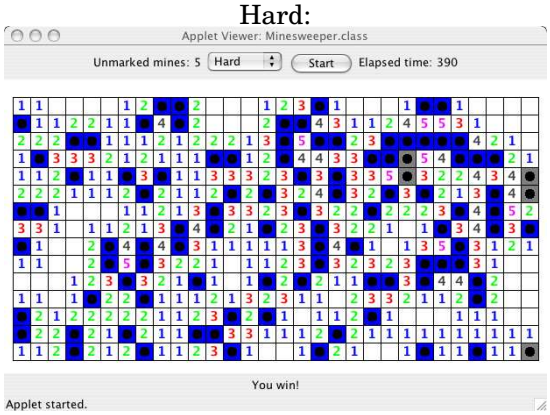
Easy:



Medium:



Hard:



- During the game, a label indicates the number of mines on the board minus the number of grid cells that have been marked.
- A timer displays the number of seconds since the start of the game. The timer stops when the player wins or loses.

- The player loses when he/she exposes a mine.
- The player wins when all the cells that do not contain a mine are exposed (regardless of how many cells are marked).
- When the game ends, the locations of all mines are displayed and a message is displayed to inform the player whether he or she has won or lost.

Implementation Details

Your program should contain the following five classes:

- Minesweeper
- Grid
- GridCell
- Level
- Timer

`Minesweeper` is the window controller. As usual, it is responsible for creating the Swing GUI components, managing user interaction with those components, handling clicks on the canvas, and initializing the game.

Since you need to be able to detect "control-clicks" in addition to regular mouse events, we have provided you with a skeleton `Minesweeper` class that includes the code to process these mouse events.

`Grid` is the two-dimensional area where the game is played. The `Grid` class should keep track of the collection of cells and should perform operations that require knowing about more than one cell. For example, the `Grid` class should determine what the neighbor count for a cell is and should decide when the game has been won or lost. Another responsibility of the `Grid` class is to translate from the x,y pixel location where the user clicks into a row,column pair to identify the grid cell that the user clicks on.

`GridCell` manages the information of an individual cell, such as whether the cell has a mine, is marked or has been uncovered. It is also responsible for doing any drawing in the cell (the gray box for an unexposed cell, the blue box for a marked cell, the number of neighboring mines for an exposed cell, etc.).

`Level` is a simple class that just allows you to group together the information for the different levels, specifically, the number of rows and columns the grid should have and how many mines should be placed in the grid.

`Timer` is an `ActiveObject`. Unlike other active objects, it does not move things around on the screen. Instead, it simply updates the `JLabel` to show the number of seconds since the game was started. Be sure to stop the timer when the game ends.

The Design

As indicated in the heading of this document, you will need to turn in a design for your Minesweeper program well before the program itself. You should include in your design a sketch of each class including the types and names of all instance variables you plan to use, and the headers of all methods you expect to write. You should write a brief description of the purpose/function of each instance variable and method.

In addition, you should provide pseudo-code for any method whose implementation is at all complicated. In particular, if a method is complicated enough that it will invoke other methods you write (rather than just invoking methods provided by Java or our library), then include pseudo-code for the method so that we will see how you expect to use your own methods.

From your design, we should be able to find the answers to questions like the following easily:

1. How do you place the mines initially on your game board?
2. What information is passed to the constructors of your classes?
3. How do you find the row and column of a grid cell on the canvas given a mouse press `Location`? (You will need to use this operation in several places.)
4. How do you determine if a cell has a mine? How do you determine if a cell is marked?
5. How do you determine how many neighbors a cell has?
6. How do you (recursively?) expose the neighbors of a grid cell that has no neighboring mines?
7. How and when is the timer updated?
8. How do you decide when a player wins or loses?

The more time you spend on the design, the easier it will be to complete the program.

Implementation Order

Begin by downloading the starter project from the handouts web page. We strongly encourage you to proceed as suggested below to ensure that you can turn in a running program. While a partial program will not receive full credit, a program that does not run at all generally receives a lower grade. Moreover it is easier to debug a program if you know that some parts do run correctly.

- Experiment with the demonstration program to familiarize yourself with the rules of the game and the user interface.
- Layout the GUI components on the main screen, but don't have them do anything initially.
- Draw an empty grid on the canvas. Work just with the Easy level (10 rows, 10 columns, 10 mines).
- Add mines to your grid. Initially, do not hide the mines so you can be sure the mines are being placed appropriately. You will find it convenient to write a method called `showAllMines` that simply draws all the mines. Until your program is working and you want to actually play the game, it would make sense to always display the mines on the grid for debugging purposes. When everything works, you can remove the call to `showAllMines`.
- Write the code to determine the neighbor count of a cell. Test this code by displaying the neighbor count in all cells that do not have a mine. Display cells with a neighbor count of 0 as empty white rectangles.
- Now hide the mines and the neighbor counts. Translate a mouse click on the canvas to a click on a grid cell. Display the contents of a cell when it is clicked on (either a mine or the neighbor count if there is no mine).
- Update the code that handles mouse clicks so that if the user clicks on a cell containing a mine, that cell's background becomes red, all the mines are displayed and the user is told that he/she lost.
- Allow an unexposed grid cell to be marked as a suspected mine location when it is "control-clicked".
- Allow a marked cell to become unexposed by a "control-click". (That is, remove the mark.)
- Update the code to handle a normal mouse click so that it does nothing if the cell is marked.

Once you have this much done, you have the basics of the game. At this point, there are a number of different things you could work on. We recommend that you do the tasks that seem simpler to you first as well as the ones worth the most points. You may find it easier to do these remaining items in a different order than we have listed them. Here are the remaining pieces of functionality to provide:

- Update the code to handle a normal mouse click to handle exposed cells differently from unexposed ones. In particular, when clicking on an exposed cell, your code should count the number of its neighbors which are marked. If this count equals the cell's neighbor count, you should expose all the unexposed neighbors.
- When a cell is exposed that has no neighbors that contain mines, automatically expose those neighbors. If any of those cells have no neighbors that contain mines, expose their neighbors as well. (This one is the hardest feature. You should probably save it for last.)
- Detect and report a win when the last unexposed cell that does not contain a mine is exposed. Note that this can happen when the last cell is clicked on directly or when it is exposed by clicking on another cell that results in exposing that cell's neighbors.
- Add and update a label that shows how many unmarked mines remain on the board. Note that if the player marks cells that do not actually contain mines, this number could become negative. At any time it should be equal to the total number of mines minus the number of marked cells.
- Add the ability to start a new game at any time.
- Allow the ability to create boards of different difficulty levels, as specified earlier in this handout.
- Add the timer.

There is a great deal of functionality to aim for in this test program. **Do not worry if you cannot implement all of the functionality**. Get as much of it working as you can. As we have done throughout the semester, we will consider both issues of correctness and issues of style when grading your program. It is always best to have full functionality, but you are better off having most of the functionality and a beautifully organized program than all of the functionality with a program that is sloppy, poorly commented, etc.

Extra Credit There are opportunities for extra credit functionality to be added to your Minesweeper program. We will give one or two points for each extension, for a maximum of 5 points extra credit. Some possible extensions are:

- Provide fields where the user can specify custom grid sizes and mine counts in addition to the three required difficulty levels.
- Draw more elaborate mines and flags to mark suspected mines. (You may play the Windows version of this game to get ideas.)
- Add an animated explosion when a mine is exposed.

Turning it in

Your design should be either neatly written or typed, and it should be turned in on paper at the beginning of class.

Submit your code in the usual way by exporting and dragging your project into the Dropoff folder for your section. Name your folder with your last name followed by TP2. Also, be sure to double check your work for correctness, organization and style.

Collaboration Guidelines

A test program is a laboratory that you complete on your own, without the help of others. It is a form of take-home exam. You may and are encouraged to consult your text, your notes, your lab work, our on-line examples, and the web pages associated with the course web page, but use of any other source (human or otherwise) for code is forbidden. You may not discuss these problems with anyone aside from the course instructors, but you should not hesitate to raise any question with them. You may only ask the TAs for help with hardware problems or difficulties in retrieving your program from a disk or network. The use of any other outside help or sources is a violation of the Honor Code.

Grading Guidelines

DESIGN AND STYLE (50 points total)

Design preparation (10 points total)

2 points Minesweeper class
3 points Grid class
3 points GridCell class
1 points Level class
1 points Timer class

Syntax Style (19 points total)

5 points Descriptive and helpful comments
4 points Good names
4 points Good use of constants
3 points Appropriate formatting
3 points Appropriate use of public/private

Semantic style (21 points total)

4 points Conditionals and loops
4 points Efficiency issues
5 points Parameters, variables, and scoping
4 points Appropriate use of arrays
4 points Appropriate methods

CORRECTNESS (50 points total)

Setup (7 points total)

3 points Layout of game screen including GUI components
2 points Display of starting grid
2 points Mines placed randomly

Basic Playability (26 points total)

2 points Mouse clicks expose grid cells
2 points Game ends when a mine is exposed
3 points Game ends when all cells without mines are exposed
2 points Exposed mined cell's background turns red
2 points All mines exposed after game ends
2 points Mines centered in cells
2 points Appropriate win/loss message displayed when game ends
3 points Neighbor count calculated correctly
2 points Display number of neighboring mines when cell exposed
2 points Numbers of neighboring mines centered in cell
2 points Mark/unmark suspected mines
2 points Marked cells cannot be exposed

Advanced Playability (6 points total)

2 points Click on exposed cell with number of marked neighbors equal to number of neighboring mines exposes all unmarked neighbors
2 points Click on 0-mine-neighbor cell displays neighborhood automatically
2 points Display of unmarked mine count maintained properly

Timer (6 points total)

2 points Timer starts when game starts
2 points Timer stops when game stops
2 points Timer increments smoothly

Game Control (5 points total)

3 points Start button starts/restarts game
2 points Difficulty levels implemented correctly