# Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Period \_\_\_\_ Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Minesweeper

In this lab, we will develop the Minesweeper game. If you have a PC running windows, you are probably already familiar with the game since it is included with windows. In this game, a grid is displayed with hidden mines. Your job is to identify the locations with mines without actually clicking on the squares that contain mines. When you choose a square, if it contains a mine then you lose! Otherwise, the square you selected along with all other surrounding squares that do not contain mines are displayed.

Note: Magic numbers in your code will earn a substantial point deduction. Use constants where appropriate. Similarly, unnecessary instance fields will earn a substantial point deduction. You should not need any new instance fields.

### Dividing up the work

A total of three objects will be used in the Minesweeper game – a MineFieldDisplay object, a Minefield object, and a Minesweeper object.

### Minefield

The Minefield object is an example of an Abstract Data Type. It encapsulates the grid that represents the minefield in the game. The nice thing about Object Oriented programming is that it is not necessary to understand the details of an Abstract Data Type in order to use it.

Internally, the Minefield maintains two arrays. One array represents the locations of all of the mines in the Minefield and the other array remembers which locations within the current Minefield been visited (either clicked on or shown to the user).

You use a Minefield object by sending messages that correspond to the public methods within the Minefield class. The Minefield has the following methods:

/\*

\* create a minefield of the specified size

\*/

public Minefield(int rows, int cols)

/\*

\* create a minefield having 9 rows and 9 cols

\*/

public Minefield()

/\*

\* get the number of rows in this Minefield

\*/

int numRows()

/\*

\* get the number of columns in this Minefield

\*/

int numCols()

/\*

\* returns true if the given location is within this Minefield

\*/

public boolean isValid(int row, int col)

/\*

\* returns true if the location contains a mine

\*/

public boolean isMine(int row, int col)

/\*

\* adds a mine at the specified location in this minefield

\*/

public void add(int row, int col)

/\*\*

\* marks an empty location as visited

\* locations containing mines cannot be marked as visited.

\*/

public void markVisited(int row, int col)

/\*

\* returns true if the specified location is marked ‘visited’

\*/

public boolean isVisited(int row, int col)

/\*

\* returns the number of mines in this Minefield

\*/

public int numMines()

### Minesweeper

Minesweeper controls the action! It encapsulates the game, and it uses a Minefield object to represent the current grid, and a MinefieldDisplay object to display the game. Most of the methods you will write will be in this class.

The Minesweeper class has the following methods:

/\*

\* constructor for Minesweeper objects

\*/

public Minesweeper()

/\*

\* method to handle button presses. When the user clicks on a location within the MineField

\* this method is automatically called. It is passed the row and column number of the location

\* that was clicked and a boolean flag that is set true if the right mouse button was used

\*/

public void pressed(int row, int col, boolean rightButton)

/\*

\* The scanField method recursively scans the field and reveals

\* all of the locations that do not contain mines

\*/

public void scanField(int row, int col)

/\*

\* The setMines method distributes the mines throughout the MineField

\*/

public void setMines(int numMines)

/\*

\* The countMines method counts the mines that are adjacent to

\* the specified location

\*/

public int countMines(int row, int col)

### MinefieldDisplay

The MineFieldDisplay object takes care of displaying the current state of the game, and interacting with the two other classes to handle mouse clicks. In order to complete the basic part of the game, you will not (and should not) modify this code. The MineFieldDisplay supports the following methods:

/\*

\* constructor for MineFieldDisplay objects

\*/

public MinefieldDisplay(Minesweeper game, Minefield field)

/\*

\* update the display

\*/

public void update()

/\*

\* show all of the locations, reveals the location of the mines

\*/

public void showAll()

/\*

\* starts the game timer

\*/

public void startTimer()

/\*

\* stops the game timer

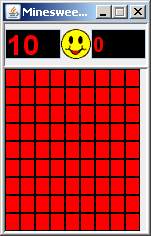
\*/

public void stopTimer()

### Getting the files

Download the Minesweeper.zip file from Athena. Choose a directory called MinesweeperName (use your name in place of Name) to hold your project, and unzip Minesweeper.zip into this directory. The zip file contains a BlueJ project, so once it is unzipped, you can run it from BlueJ. Using BlueJ, click on “Project -> Open Project...” and navigate to the folder you just created. When you open the project, you will see a display of the three major classes and the Main class.

Compile the project and run it. You should see the Minesweeper display.



The game won’t do much at this point, but you can click on the squares within the Minefield.

### Completing the Minefield class

Add your name and the current date. Here is how you should do this. Leave the original author and date. Near the bottom of the documentation at the top of the code, add your information is a line that looks like:

2009-12-01 Susan King Added the primary code to the methods isValid and numMines.

You will not need to add any instance fields to this class. **Do not add any instance fields to this class. A point deduction of 100% will be made for any additional instance fields you put in this class.**

Complete the method isValid(int row, int col) which returns true iff the location specified by row and col is within the grid. You will need to use the **accessors** numRows() and numCols(). Remove the line of code that throws an exception.

Complete the method numMines() which returns the total number of mines contained in this Minefield. You will need to use the **accessors** numRows() and numCols(). Remove the line of code that throws an exception.

**Completing the Minesweeper Class**

Add your name and the current date. Here is how you should do this. Leave the original author and date. Near the bottom of the documentation at the top of the code, add your information is a line that looks like:

2009-12-01 Susan King Added the primary code to the methods setMines, countMines, pressed, scanField.

You will not need to add any instance fields to this class. **Do not add any instance fields to this class. A point deduction of 100% will be made for any additional instance fields you put in this class.**

##### The setMines Method

This method is responsible for placing the mines randomly in the Minefield. To do this, we will use the random number generator from the Math class. The message Math.random() returns a random double in the range [0.0, 1.0).

In order to place the mines, you need an integer between 0 and the number of rows and an integer between 0 and the number of columns. Suppose you want a random integer between 0 and 6. The following code will accomplish this task:

int randomNumber = (int)(Math.random() \* 6);

In order to place the mines, you will need to generate a random row number and a random column number. Then you will need to check to see if there is already a mine in that location. If there is no mine, place on using the appropriate method in the Minefield object. Otherwise, you must generate another random row and column number. Repeat the process until all of the mines have been placed.

You might want to test your code by printing out the location of each mine that you place.

##### The countMines Method

This method accepts a location as a row number and a column number, then counts the number of mines adjacent to this location. There may be as many as eight locations to check. When writing this method, be sure not to try to access any invalid locations!

Write the countMines method. If you printed out the locations of the mines above, you can devise a test for this method. Testing your code as you go is recommended (unless you like to spend many hours debugging).

#### *The pressed Method*

This method is automatically called by the MinefieldDisplay class whenever a location within the Minefield is clicked. Three parameters are passed to the method: the row and column number of the location that was clicked, and a boolean variable that indicates whether or not the right mouse button was used. Add the following line of code to the pressed method:

System.out.println("row = " + row + " column = " + col +

" rightButton = " + rightButton);

Test your program.

Now, you can write the code that responds to the mouse clicks. When a mouse click occurs, you need to perform the following:

*If the location contains a mine, then the game is over. Send the showAll method to the display object to end the game. Otherwise, use the scanField method to scan all of the adjacent squares, revealing empty squares. Then send the update method to the display object.*

You won’t be able to fully test this code until you finish.

##### The scanField Method

The scanField method uses a technique called ‘recursive backtracking’ to reveal the mine field up to the boundaries of individual mines. To write this method, you need to develop and comment the base case. Then, in the recursive step, you need to scan the field in all possible directions. Be sure that your base case includes a test for a valid location, and that you mark each location you visit as visited. Otherwise, the recursion will not end.

When you complete the scanField method, you will have an operational game!

### To score above 90%

You must do both of the following to earn 100%:

1. Change the smiley face to a frowny face when you lose the game.

2. Add right mouse button processing, including flags