UNBC Laboratory Assignment
Computer Science 101—Winter 2008

Cross Word Grid

Due Date:

This assignment is due Wednesday 2008-03-31.

Purpose:

To become familiar with the `javax.swing.*` architecture for using `Graphics` to produce graphical output.

Reading Assignment:

Read Chapter 14 of the textbook before attempting this assignment.

Cross Word Grid

Write a "cross-word" generating program that takes input similar to that shown in Figure 1, and produces output similar to that shown in Figure 2.

The program should read from an input file and produce a GUI.

The format of the input file is illustrated in Figure 1. The first two lines of an input file are numbers: the first number is the number of rows (r) in the crossword and the second number is the number of columns (c). After that come r lines of text, each line containing exactly c-characters. The last r lines of text consist only of alphabetic characters, all of which are either lower-case or capital `'X'`.

The format of the output GUI is illustrated in Figure 2. It consists, at least conceptually, of three components.

The first component is an $r \times c$ grid of squares that are either black or white. `X`s on the input are represented by black squares on the output. Other characters are drawn in white squares in uppercase. White squares have a small number in the top-left corner if they are the beginning of either a horizontal or vertical word. (It is possible that they may be both. See the square numbered 4 in the example.) One-letter words are ignored (in the example, the A in the top row is not considered a vertical word).

The large letters should be centered if possible. (See below.)

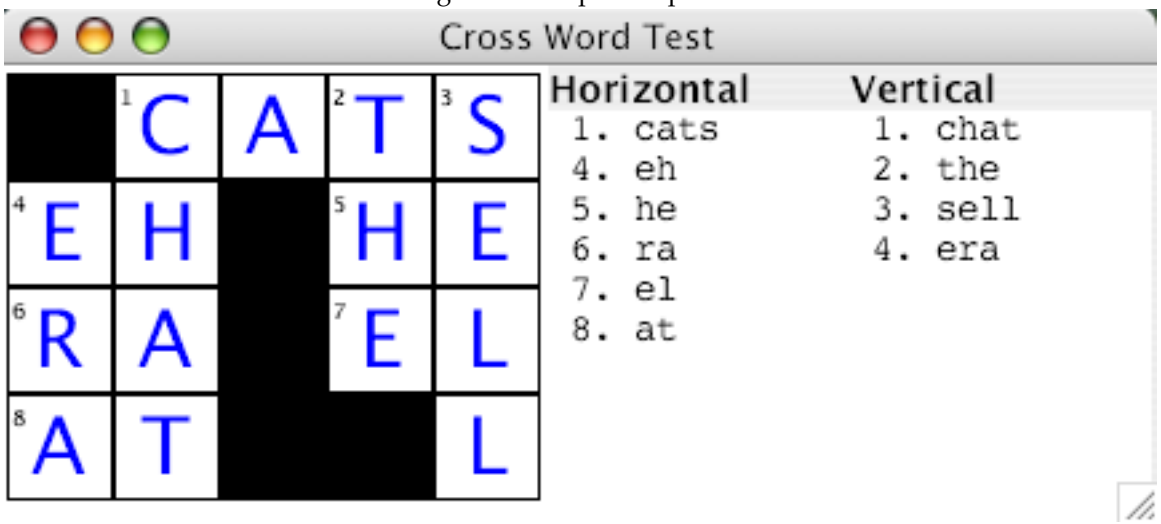
The second component is a column of the horizontal words with their starting numbers. The third component is a column of the vertical words with their starting numbers.

Figure 1: Sample Input

in.txt

```
4
5
Xcats
ehXhe
raXel
atXXl
```

Figure 2: Sample Output



Programming Details

The following advice is only recommended advice. If you find other ways to implement the functionality requested, please feel free to do so.

The second and third components of the display are most easily implementable as `JTextAreas`. To use a different font for the titles, make the title a `JLabel`, and combine it with the word list using a `JPanel` with a `BoxLayout layout`.

The grid in the first component is the hardest component to display. One approach that is possible here is to subclass `JPanel`, and then use `setLayout(null)` to remove the layout manager, and then explicitly set the location of each object that you wish to display, à la

```
add(mySubObject) ;
mySubObject.setLocation(x, y) ;
```

To get the grid itself to take up enough space in the overall frame, you may need to override the `getMinimumSize` method.

The individual cells of the can be subclassed from `JComponent`. If you use `setBorder`, it is easy to get the black line around them. In this case you want to override the

```
protected void paintComponent(Graphics g)
```

method in order to the paint the cell. Remember to call `super.paintComponent(g)` first, and then do whatever painting you need. You probably want to do a `g.setColor` one or more times, a `g.setFont` zero or more times, and then various `g.fillRect`'s and `g.drawStrings`. The graphics object that you get *has co-ordinates relative to the cell, not the whole frame*.

To center characters in a cell, you will want to use a `FontMetrics` object.

```
FontMetrics fm = graphics.getFontMetrics(getFont()) ;
... fm.charWidth(myChar);
... fm.getStringBounds(""+myChar,0,1,arg0).getHeight() ;
... fm.getDescent()
```

The `FontMetrics` object is designed for getting measurements of a character shape. The *descent* of a font measures how much characters can fall below the baseline. The third argument of the `graphics.drawString(s,x,y)` function specifies the baseline for the string, not the bottom of its bounding rectangle.