Solving Word Search Puzzles Using SAS®
Paulette W. Staum, Paul Waldron Consulting, West Nyack, NY

ABSTRACT
Have some fun solving word search puzzles using SAS®. First, we will load the puzzle and gather information about it. Next, the puzzle will be stored in an array and searched for the target words using character functions. Finally, we’ll display the solution, including circled answers, using the Data Step Graphics Interface.

INTRODUCTION
Solving word search puzzles using SAS demonstrates a variety of useful techniques from data input, through arrays and string manipulation, to final output of the solution. An example of a word search puzzle is shown below. Words can be horizontal, vertical or diagonal. Words can start at the edge of the puzzle or in the interior of the puzzle. Horizontal words can start at the left or the right. Vertical words can start at the top or the bottom. You can probably see some words in the puzzle already.

STEP 1 – LOAD THE PUZZLE AND WORD LIST
People often solve word search puzzles without a list of the words to find, but this program is not that smart. Both the puzzle grid and a list of words are required inputs. The first step is to input the puzzle grid and the word list into SAS data sets. You can create a word search puzzle by using a SAS program in a SUGI paper by Robert Matthews or by visiting a web site that lets you create word search puzzles.

data puzzle;
  input line $ 1-80;
  line = upcase(compress(line));
datalines;
WERADPIMI
UNESUGNSQ
MSGIATPIA
UIELZZUPL
OCKAAHTMI
TONOMREVVN
ASQLXCWSD
SADROWHEE
ESCIARRAYX
; run;

data terms;
  input term $ 1 - 64;
  term=upcase(compress(term));
datalines;
NESUG
Word
Search
Puzzle
Vermont
Autumn
color
array
SAS
SQL
Index
Input
DSGI
; run;
STEP 2 – GATHER METADATA ABOUT THE PUZZLE

Since we would like to be able to solve any word search puzzle of reasonable size, let’s give our program some flexibility. We’ll want to know the number of rows and columns in the grid, the number of words in the word list, the length of the longest string in the puzzle, the length of the longest word and the total number of characters.

```
* Get number of rows and columns in the puzzle, number of words in list, and lengths of longest string and longest term;
proc sql noprint;
   select count(*), max(length(line)) into :NRows, :NCols from work.puzzle;
   select count(*), max(length(term)) into :NTerms, :MaxLenTerm from work.terms;
quit;

%* left-justify new macro variables;
%let NRows  = %left(&NRows);
%let NCols  = %left(&NCols);
%let NTerms = %left(&NTerms);
%let MaxLenTerm = %left(&MaxLenTerm);

%let MaxLenToSearch = %sysfunc(max(&NRows,&NCols));  /* longest puzzle string*/
%let NTotChars = %sysfunc(&NRows*&NCols);   /* number of characters in puzzle*/
%let puzzle=NESUG;                      /* Puzzle name for title line*/
```

STEP 3 – STORE THE PUZZLE AND WORD LISTS IN ARRAYS

Now it is time to store the puzzle in a form that will be simple to process. The word search process is inherently two-dimensional, with rows and columns, so let’s load the puzzle into a two-dimensional array. Note that this is done at the beginning of one DATA step that will include all the code in both steps 3 and 4.

```
data solution;
   * Read the puzzle, one line at a time;
   set puzzle end=eofpuzzle;

   * Set up two-dimensional array to hold puzzle;
   length char1-char&NTotChars $1;
   retain char1-char&NTotChars;
   array puzzle {&NRows, &NCols} char1-char&NTotChars;

   * Store each character of each puzzle line in two-dimensional array;
   iRow = _n_;
   do iCol = 1 to &NCols;
      puzzle{iRow, iCol} = char(line,iCol);
   end;
```

The list of words should be loaded into a one-dimensional array. Here it is read by a loop which executes during the first iteration of the DATA step.

```
* Load terms once - when first line of puzzle is loaded;
length term1-term&NTerms $&MaxLenTerm;
array terms {&NTerms} term1-term&NTerms;
retain term1-term&NTerms;
if _n_=1 then do iTerm = 1 to &NTerms;
   set terms;
   terms{iTerm} = term;
end;
```
STEP 4 – FIND THE WORDS

After the puzzle and the word list are both loaded, it’s time to look for the words in the puzzle grid. This can be done at the end of the DATA step that reads the puzzle and word list. There are three parts to finding the words:

1. Build all the text strings in the puzzle grid. In other words, concatenate the characters in the puzzle array for rows, columns and diagonals.
2. Examine each string to see if any words (or reversed words) are included in the string.
3. Whenever a word is found, output an observation with the word, the starting row and column, the direction and a “reverse flag”.

One way to write this section of the program would be to have a separate section for each possible text string direction, but there is another, better approach suggested by Paul Dorfman’s code in the SUGI code in the SUGI paper on creating word search puzzles by Robert Matthews. This code defined an array with entries for each of the directions in the puzzle grid – horizontal, vertical, and diagonals.

The array contained numbers (e.g., 1, 0, -1) describing how the row and column indices should be incremented when concatenating characters into a text string. For example, a horizontal row can be built by incrementing the column index, while leaving the row counter unchanged. A vertical column can be built by incrementing the row index, while leaving the column counter unchanged. One type of diagonal (starting in the top left corner) can be built by incrementing both the row and column indices. Another type of diagonal (starting in the bottom left corner) can be built by decrementing the row index from its maximum while incrementing the column index.

This powerful technique allows any program with directional processing of a grid to be much more compact. For our purpose, this can be extended by defining arrays with additional characteristics of directions, for example:

1. Text description
2. Starting and ending rows
3. Starting and ending columns
4. Row and column offsets for moving to additional text strings
5. Row and column increments when building a text string

* Notice repetition of diagonal directions for upper / lower halves of puzzle;
* Some arrays have the same value for each entry and could be omitted;

array dirdesc {6} $8     ('W_E  ' 'N_S  ' 'NW_SE' 'NW_SE' 'SW_NE' 'SW_NE');
array RowStart       {6} (  1      1         1       1     &NRows
                       &NRows
                    );
array ColStart       {6} (  1      1         1       1     1      1 );
array RowStartOffset {6} (  1      0         0       1    -1     0 );
array ColStartOffset {6} (  0      1         1       0    0      1 );
array RowEnd         {6} ( &NRows &NRows &NRows &NRows   1      1 );
array ColEnd         {6} ( &NCols &NCols &NCols &NCols &NCols  &NCols
                       &NCols
                    );
array StringRowIncr {6}  (  0      1         1       1   -1     -1 );
array StringColIncr {6}  (  1      0         1      1    1     1 );

On the next page, there is code using these characteristics to build text strings for half of the directions. It shows how to process directions with RowStartOffset not equal to 0, in other words, directions where adjacent text strings differ in their starting rows. These include horizontal rows and half of the diagonals.

Vertical columns and the other half of the diagonals have ColStartOffset not equal to 0. For them, adjacent text strings differ in their starting columns. The complete program has similar code for this case which is omitted from this paper.

After each text string is built, it is searched for the presence of words from the word list.
length direction $12;

if eofpuzzle then do; * Solve the puzzle!;

do iTYPE = 1 to 6; * loop through horizontal, vertical and diagonal strings;
direction = dirdesc{iType};

* Note that either RowStartOffset or ColStartOffset is zero;
* If the row starting points vary, loop through starting points;
if RowStartOffset{iType} NE 0 then
  do StartRow = RowStart{iType} to RowEnd{iType} by RowStartOffset{iType};
  ToSearch='';      * start with empty text string;
iRow=StartRow;
  do iCol = ColStart{iType} to ColEnd{iType} by StringColIncr{iType};
    if (iRow > 0 and iRow <= &NRows) and (iCol > 0 and iCol <= &NCols)
      then ToSearch = cats(ToSearch,puzzle{iRow, iCol});
    end;
  end;
end;

* Now that we have the puzzle text string, look for words/terms in it;
do iTerm = 1 to &NTerms;
  MatchPos = index(ToSearch,trim(terms{iTerm}));
  * Look for reversed word;
  RMatchPos = index(left(reverse(ToSearch)),trim(terms{iTerm}));

  * Switch to beginning for reversed words, and flag reversal;
  if RMatchPos > 0 and MatchPos=0 then do;
    MatchPos = length(ToSearch) - RMatchPos + 1 ;
    ReverseFlag=1;
  end;
  else ReverseFlag=0;

  * If word found, output word with starting row and column;
  if MatchPos > 0 then do;
    Term = Terms{iTerm};
    LenTerm = length(term);
    TheRow=StartRow + (MatchPos-1) * StringRowIncr{iType};
    TheCol=MatchPos * StringColIncr{iType};
    output;
  end;
  end; * changing row of starting point (rowstartoffset{itype} ne 0);

else if ColStartOffset{iType} NE 0 then
  * Similar code for directions with changing starting columns goes here;
end;
end; * do itype=1 to 6;
end; * eofpuzzle;
run;
The output from the DATA step is a complete solution data set with variables for the word, the starting row and column, the direction and a reverse flag. Directions are identified as:

- west-east (W_E)
- north-south (N_S)
- northwest-southeast (NW_SE)
- southwest-northeast (SW_NE).

After removing duplicate words, the output solution data set looks like this:

<table>
<thead>
<tr>
<th>Term</th>
<th>Row</th>
<th>Col</th>
<th>Direction</th>
<th>Reverse Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td>9</td>
<td>4</td>
<td>W_E</td>
<td>0</td>
</tr>
<tr>
<td>AUTUMN</td>
<td>1</td>
<td>4</td>
<td>NW_SE</td>
<td>0</td>
</tr>
<tr>
<td>COLOR</td>
<td>5</td>
<td>2</td>
<td>NW_SE</td>
<td>0</td>
</tr>
<tr>
<td>DSGI</td>
<td>1</td>
<td>5</td>
<td>SW_NE</td>
<td>1</td>
</tr>
<tr>
<td>INDEX</td>
<td>5</td>
<td>9</td>
<td>N_S</td>
<td>0</td>
</tr>
<tr>
<td>INPUT</td>
<td>1</td>
<td>7</td>
<td>N_S</td>
<td>0</td>
</tr>
<tr>
<td>NESUG</td>
<td>2</td>
<td>2</td>
<td>W_E</td>
<td>0</td>
</tr>
<tr>
<td>PUZZLE</td>
<td>4</td>
<td>8</td>
<td>W_E</td>
<td>1</td>
</tr>
<tr>
<td>SAS</td>
<td>7</td>
<td>2</td>
<td>N_S</td>
<td>0</td>
</tr>
<tr>
<td>SEARCH</td>
<td>3</td>
<td>2</td>
<td>NW_SE</td>
<td>0</td>
</tr>
<tr>
<td>SQL</td>
<td>7</td>
<td>2</td>
<td>W_E</td>
<td>0</td>
</tr>
<tr>
<td>VERMONT</td>
<td>6</td>
<td>7</td>
<td>W_E</td>
<td>1</td>
</tr>
<tr>
<td>WORD</td>
<td>8</td>
<td>6</td>
<td>W_E</td>
<td>1</td>
</tr>
</tbody>
</table>

**STEP 5 – DISPLAY SOLUTION WITH DATA STEP GRAPHICS INTERFACE**

When people find a word in a word search puzzle, typically they circle the word. Can SAS mimic this type of word identification? One way is to use the DATA Step Graphics Interface. This is a handy tool for creating customized graphical output using a DATA step. The graphic is stored as a catalog entry. Our basic tools will be the DSGI functions GINIT, GSET, GRAPH and GDRAW.

- **GINIT** initializes the DSGI environment.
- **GSET** sets characteristics of the DSGI environment, for example defining a window size and setting text font and height.
- **GRAPH** manages graphics segments in catalogs
- **GDRAW** draws text, lines, shapes, symbols or images
The following code creates a graphic segment with a picture of the puzzle grid of characters

```sas
data _null_;  
set puzzle end=eof;  
if _n_=1 then do;  
  * Initialize DSGI;  
  rc=ginit();  
  * Set size for window #1:  
  rc=gset('window',1,0,0,100,100);  
  * Activate window #1;  
  rc=gset('transno',1);  
  * Delete any old graphics segment with the name MyGraph in current catalog;  
  * Otherwise, a numeric suffix will be included in the name of the new graph;  
  rc=graph('delete','Mygraph');  
  * Create a new graphics segment with the name MyGraph;  
  rc=graph('clear','Mygraph');  
  * Set text font and height;  
  rc=gset('texfont','SWISSBU');  
  rc=gset('texheight',3);  
  * Draw title as text string at x, y coordinates;  
  * x = 100 for right edge of page, y = 100 for top of page;  
  rc=gdraw('text', 4, 95 , "puzzle Puzzle Solution");  
end;  
* Display each line of characters from input puzzle;  
do i = 1 to length(line);  
  rc=gdraw('text', 1 + 2*(i), 100 - 3 * _n_ - 6, substr(line,i,1));  
end;
```

Now that the puzzle has been displayed, next let's display each target word in color and circle it. We'll read the puzzle solution data set to tell us how to place circles (ellipses) around the words that we found. Also, just for fun, we'll set the color according to the direction of the word, e.g., brown for horizontal left-to-right words. The new colored letters will replace the black letters displayed above as part of the grid. We will need some additional DSGI syntax:

- **GDRAW(ELLIPSE,...)** draws an ellipse. It has 7 arguments:
  - x coordinate of ellipse position
  - y coordinate of ellipse position
  - major axis length
  - minor axis length
  - starting angle
  - ending angle
  - angle of major axis relative to 3:00

- **GSET(TEXCOLOR,colormum)** changes the color of text.
We'll need to calculate the x and y coordinates and the length of each word. The code below shows an example for the simplest case, a left-to-right horizontal word. (Note that this DATA step uses StartCol and StartRow differently from the solution DATA step.)

```plaintext
if eof then do;
  * Read the solution data set once for each word.
  A word could be in the puzzle (and solution data set) more than once.;
  do iTerm = 1 to &NSolutions;
    set solution;
    * Variables in the solution data set are term, direction, startrow and startcol. We will need the length of the word also;
    pointer=substr(direction,1,3);
    lenterm=length(term);
    StartRow=TheRow; StartCol=TheCol;
    if pointer='W E' and ReverseFlag=0 then do;
      do iChar = 1 to LenTerm;
        * Draw each character in the word in color;
        rc=gset('texcolor',10); * Use brown=10 for horizontal terms;
        rc=gdraw('text', 1 + 2*(startcol + ichar - 1),
                  100 - 3 * startrow -6,
                  substr(term,iChar,1) );
      end;
    * Draw ellipse around term;
      middlecol = 1 + (2 * (startcol + .5*lenterm ));  %* x;
      middlerow = 100 - (3 * (startrow ) ) - 4.5;  %* y;
      rc = gdraw('ellipse', middlecol, middlerow,
                  2*(lenterm), 2,
                  0, 360, 0);
    end;
  end;
  rc=gterm();
end;
* Close graphics segment;
rc=graph('update');
* End DSGI;
rc=gterm();
end;
run;
```

After similar code for other directions, close the graphic production and end the DATA step.
Here is the output. It isn’t beautiful, but it is reasonably effective. The ‘circling’ location is approximate, not exact and in some cases (e.g., “PUZZLE”) the word is not a uniform color, because the same letter is used in more than one word.

CONCLUSION

The flexibility and power of SAS are demonstrated in this automated solution for word search puzzles using:

- INFILE and INPUT statements for reading puzzles
- PROC SQL select ... into macro variables for simple generalization of code to different size puzzles
- Two-dimensional array processing for extraction of strings that may contain target words
- String processing functions for matching target words to the strings in the puzzles
- DATA Step Graphics Interface for displaying the solution complete with color and circling of target words

REFERENCES

www.armoredpenguin.com is one of several sites that generates word search puzzles.


ACKNOWLEDGMENTS

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Many other methods could be used for this task. If you have suggestions, or would like a copy of the completer program, please contact me at:

Paulette W. Staum
Paul Waldron Consulting
2 Tupper Lane
West Nyack, NY 10994
staump@optonline.net