Fall 2013 CSCI2961: Programming in Python Homework 3

General Instructions

This homework assignment is to be completed individually. Do not share code or review anyone else's code. Work on this assignment is to be your own.

Submit your homework via RPILMS before midnight on the due date. Put all of your code into exactly one Python file and name it with the homework number followed by an underscore, followed by your RCS userid. For example, if your RCS userid is mehtaa2, then your Python file name for this homework assignment must be hw3_mehtaa2.py.

Be sure to comment your code and include your name at the top of each file submitted.

Word Games

Valid Words:

First download the "ospd.txt" (Official Scrabble® Player's Dictionary) file from EX04_Sep17 if you have not already done this, to use for development.

generateWordFindPuzzle:

Write a Python function called `generateWordFindPuzzle()` that takes exactly four arguments (in this order): the dictionary of valid words (as a list); the number of words to hide in the puzzle; the number of rows in the puzzle; and the number of columns in the puzzle. This function generates a word-find puzzle (ALL UPPERCASE) and return the puzzle as a string (see below for the additional return value). Note that the string must contain newline characters (`'\n'`) at the end of each line (except for the last line). An example puzzle (with words PYTHON and PERL) is:

```
QOFJEFUEJQ
SWEOPYTHON
WEJDEJWVJE
WEFORWOEFF
WJEELVNMZM
PFEPEPDDAE
```

Randomly select words from the dictionary of valid words, then hide these words in the puzzle by placing them in random locations and in random directions. Words may appear in the horizontal, vertical, and diagonal directions and may be either forwards or backwards. Fill in all empty spaces with random letters.

In addition to returning the puzzle as a string, the `generateWordFindPuzzle()` function must return a tuple of the hidden words (which must be in sorted order). Therefore, this function has exactly two return values (a string and a tuple), as in:

```
puzzle =
'QOFJEFUEJQ\nSWEOPYTHON\nWEJDEJWVJE\nWEFORWOEFF\nWJEELVNMZM\nPFEPEPDDAE'
wordlist = ( 'PERL', 'PYTHON' )
return puzzle, wordlist
```

findWords:

Next, write a Python function called `findWords()` that takes exactly two arguments (in this order): the dictionary of valid words (as a list); and the puzzle (as a string). The function must return a tuple of all valid words found.

In general, if a puzzle generated using the `generateWordFindPuzzle()` function above with `N` hidden words is given as input to the `findWords()` function, a tuple of at least size `N` should be returned from `findWords()`. Be sure that this tuple is sorted and contains no duplicate words.

generateDoubleWordSquare:

A double word square is a grid of letters with the same number of rows and columns in which all words across and down are valid. For example, a valid 5x5 double word square is:

```
SCENT
CANOE
ARSON
ROUSE
FLEET
```

Write a Python function called `generateDoubleWordSquare()` that takes exactly two arguments (in this order): the dictionary of valid words (as a list); and the size of the double word square (e.g. 5). Using random words from the list, this function must generate a valid double word square (ALL UPPERCASE) and return it as a string. Note that the string must contain newline characters (`'\n'`) at the end of each line (except for the last line).

Additional Guidance

Use whatever approach you like (e.g. recursion, brute force, genetic algorithm, etc.) for the algorithms for the word search.

You can add other functions as needed to organize your solution (for example, to load a dictionary of words and return it), but make sure you include the functions with the specified signatures above, to facilitate grading.

Program startup

The program should be submitted as a module that can be imported (when imported as a module, only the functions should be defined; there should be no output).

The module should also be able to be invoked as a command line utility, similar to the way UNIX shell commands (e.g. “ls”, “cat”, …) are invoked. When invoked this way, your program should be able to generate either a word find puzzle or a double word square (depending on the arguments supplied). You should format and print the results on the standard output, when your program is invoked this way.

Read the documentation for the argparse standard Python module, and use that module as it is intended, to manage your command line parsing/processing. What this means is that you should support argument parsing, but you should architect it in a way that minimizes the amount of hard coded parsing code that you write (let the argparse module do the ‘heavy lifting’, and accept the formatting and behavior that is provided for you by the argparse module for argument specifications, help/usage output, etc.).

At a minimum, you should support the following options; you can pick reasonable defaults for the parameters, in case the user does not specify a value:

```
-h, --help        show the help message and exit
-f --file FILE    Load the specified dictionary file, and use it as the
                  source of words
-w --nwords NWORD Specifies the number of words to generate
-r --rows NROW    Specifies the number of rows to generate
-c --columns NCOL Specifies the number of columns to generate
-s --square SIZE  Generates a double word square of this size
```

The “-s --square" argument is the only one that will cause a double word square to be generated; otherwise, a word find puzzle is generated.