Project 1a: Hangman

CS Y11

February 18, 2019

CLICK HERE FOR FILES NEEDED

1 Problem Introduction

For this assignment, your mission is to write a program that plays the game of Hangman. The program is designed to give you some practice writing programs that manipulate strings and files.

When the user plays Hangman, the computer first selects a secret word at random from a list built into the program. The program then prints out a row of dashes one for each letter in the secret word and asks the user to guess a letter. If the user guesses a letter that is in the word, the word is redisplayed with all instances of that letter shown in the correct positions, along with any letters correctly guessed on previous turns. If the letter does not appear in the word, the user is charged with an incorrect guess. The user keeps guessing letters until either (1) the user has correctly guessed all the letters in the word or (2) the user has made seven incorrect guesses. Two sample runs that illustrate the play of the game are shown in Figure 1 on the next page.

2 Part 1

In the first part of this assignment, your job is to write a program that handles the user interaction component of the game everything except the graphical display. To solve the problem, your program must be able to:

1. Choose a random word to use as the secret word. That word is chosen from a word list, as described in the following paragraph.
2. Keep track of the user's partially guessed word, which begins as a series of dashes and then gets updated as correct letters are guessed.
3. Implement the basic control structure and manage the details (ask the user to guess a letter, keep track of the number of guesses remaining, print out the various messages, detect the end of the game, and so forth).

The only operation that is beyond your current knowledge is that of representing the list of words from which you can choose a word at random. For the first two parts of the assignment, you will simply make use of a method that we've given you called getRandomWord(). The implementation of the method you've been given is only a temporary expedient to make it possible to code the rest of the assignment. In Part II, you will replace the definition weve provided with one that utilizes a list of words from a data file.

The strategy of creating a temporary implementation that provides enough functionality to implement the rest of the program is a common technique in programming. Such temporary implementations are usually called stubs.A game that used this implementation of getRandomWord would quickly become uninteresting because there are only ten words available. Even so, it will allow you to develop the rest of the program and then come back and improve this part later.

2.1 Clarification

Part I is a string manipulation problem using the sample runs in Figure 1 should be sufficient to illustrate the basic operation of the game, but the following points may help to clarify a few issues:

1. You should accept the user's guesses in **either lower or upper case**, even though all letters in the secret words are written in upper case.

2. If the user guesses something other than a single letter, your program should tell the user that the guess is illegal and accept a new guess.

3. If the user guesses a correct letter more than once, your program should simply do nothing. Guessing an incorrect letter a second time should be counted as another wrong guess. (In each case, these interpretations are the easiest way to handle the situation, and your program will probably do the right thing even if you don't think about these cases in detail.).

Remember to finish Part I before moving on to Part II.

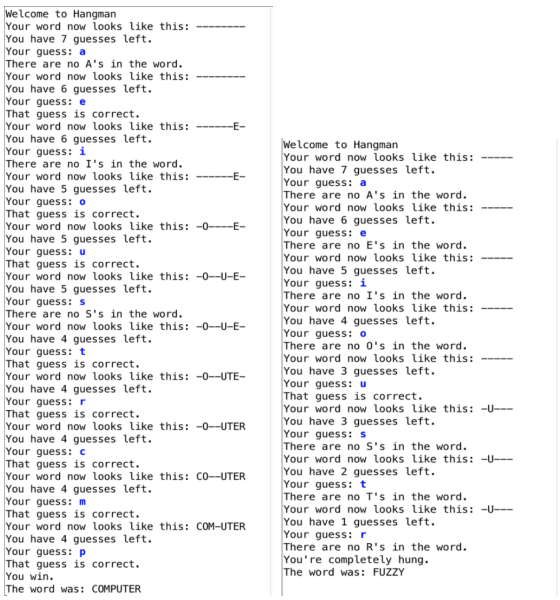

Figure 1: Sample runs

3 Part II

Your job in this part of the assignment is simply to re-implement the getRandomWord method so that instead of selecting from a meager list of ten words, it reads a much larger word list from a file. The steps involved in this part of the assignment are as follows:

1. Open the data file HangmanLexicon.txt using a Scanner that will allow you to read it line by line.

2. Read the lines from the file into an ArrayList.

3. Reimplement the getRandomWord method so that it uses the ArrayList from step 2 as the source of the words.

   The first two steps should be at the start of your program in their own method. getRandomWord should rely on the file reading having already happened; if you were to call getRandomWord twice, you would not read the file twice. Note that methods which use getRandomWord should not have to change in response to this change in the implementation. Insulating parts of a program from changes in other parts is a fundamental principle of good software design.

4 Possible Extensions

1. Allow the user to play multiple games.

2. Once you get the basic structure working, you could expand the program to play something like Wheel of Fortune, in which the single word is replaced by a common phrase and in which you have to buy vowels

3. You could write an A.I. agent that plays the game for the user. One common strategy is to guess the most frequent letter. After loading words from the lexicon, find out which letters come up most often.

4. Anything you think would be great or that you came up with through receiving feedback from others!

Acknowledgments to Stanford's CS106A class for this project idea and for some of the materials.